MASTER EN SCIENCES MATHÉMATIQUES

Le manipulateur de séries MSNam

HENRARD, Joffroy

Award date:
2013

MASTER EN MATHÉMATIQUES

Le manipulateur de séries MSNam

Joffroy Henrard

2013

Université de Namur

Promoteur : Anne Lemaître

Directeur : Anne-Sophie Libert

Le manipulateur de séries MSNam

Joffroy Henrard

Namur, le 27 mai 2013

Remerciements

Je profite de l'opportunité qui m'est laissée pour remercier tout ceux qui ont rendu possible la rédaction de ce manuscrit.

Je tiens tout d'abord à remercier Anne Lemaître, mon promoteur, pour avoir accepté de superviser ce travail ainsi que pour son soutien et ses conseils. Je la remercie également pour la curiosité qu'elle a éveillé en moi lors de ses cours, éveillant ainsi en moi le goût de la recherche et de la mécanique céleste.

Je tiens tout particulièrement à exprimer ma reconnaissance envers Anne-Sophie Libert, mon directeur, pour son aide, ses remarques, ses conseils et sa disponibilité. Je la remercie également pour l'intérêt et la motivation dont elle a fait part tout au long de ces deux années.

Je remercie également Antonio Giorgilli pour avoir accepté d'être mon maître de stage pendant ces trois mois à Milan. Je le remercie également, ainsi que Marco Sansottera pour leur dynamisme et pour m'avoir permis d'utiliser leurs outils et ainsi découvrir d'autres facettes de la mécanique céleste.

J'adresse également mes remerciements à Timoteo Carletti et à Audrey Compère qui ont contribué au travers de leurs conseils à la rédaction d'une partie de ce travail.

Merci également à André Füzfa pour m'avoir donné goût à la programmation, à Benoît Noyelles et à son incomparable façon d'aborder la dynamique céleste.

J'adresse aussi mes remerciements à tous les professeurs et assistants namurois que je n'ai pas cités plus haut pour leur disponibilité et leur implication tout au long de mes études.

Un grand merci aussi aux secrétaires du département, Pascale Hermans pour ses informations en continu et Martine Van Caenegem pour son aide en Latex.

Un merci tout particulier aussi à mes camarades, toutes années confondues, pour leur soutien et la bonne humeur journalière lors des temps libres.

Enfin, merci à ma famille qui, malgré les moments difficiles, a toujours cru en moi et m'a soutenu dans un domaine où ils n'ont jamais pu m'aider.

A tous et à ceux que j'ai oubliés, merci encore !

Le manipulateur de séries MSNam
par Joffroy Henrard

Les récentes avancées dans le domaine des systèmes dynamiques requièrent l'utilisation de nombreux outils informatiques. Il existe beaucoup de ces outils pour le traitement de données symboliques, notamment Maple ou Mathematica.

Nous avons la chance de posséder notre propre manipulateur symbolique à Namur, créé par Jacques Henrard il y a de cela quelques années. Utilisé dans le cadre de la mécanique céleste, dans le cas précis du traitement des séries de Poisson, celui-ci est peu documenté et malheureusement peu utilisé.

Le but de ce travail sera d'une part de documenter entièrement le manipulateur, rendant son utilisation plus accessible, et d'autre part de le compléter. De plus, nous l'utiliserons pour résoudre des problèmes concrets et nous le comparerons également avec un de ses concurrents.

Nous espérons que ce travail permettra de faire comprendre l'intérêt de conserver ce manipulateur dans le patrimoine de notre Université mais aussi, et surtout, d'attirer de nouveaux utilisateurs potentiels pour des applications et des améliorations futures.

The series manipulator MSNam
by Joffroy Henrard

Recent progresses in dynamical systems require the use of many computer tools. Lots of these tools exist to deal with symbolic data, as for example Maple or Mathematica.

We are fortunated to have our own symbolic manipulator in Namur, built by Jacques Henrard a few years ago. Usefull in celestial mechanics, precisely to deal with Poisson series, its documentation is obsolete and it is used too few.

On the one hand, in this work, we are going to make an entire documentation of the manipulator, making it more accessible, and on the other hand we are going to complete it. Moreover, we are going to use it for solving concrete problems and we are also going to compare it with one of its concurrents.

We hope that this work will show the interest to keep this manipulator in our University's heritage but especially draw new potential users for future applications and improvements.

Table des matières

Introduction

Les hommes, plus particulièrement les astronomes, observent le ciel depuis la nuit des temps. En utilisant des instruments rudimentaires et des théories de calcul très simples, ils ont mesuré toutes sortes de distances. D'abord celle entre deux points sur Terre, puis la distance Terre-Lune et même celle entre Mars et la Terre. La plupart de ces procédés ne nécessitent que quelques notions de géométrie, un peu d'observation et beaucoup d'imagination. Au fil des siècles, la compréhension du monde et de l'espace qui nous entoure s'est améliorée. La Terre plate est devenue ronde, le géocentrisme qui suppose que la Terre est immobile au centre de l'univers est devenu héliocentrisme avec notre Terre qui tourne autour du Soleil, et les cercles décrits par les astres sont devenus des ellipses.

Pour en arriver à de telles conclusions, les astronomes des temps modernes tels que Copernic, Galilée ou Kepler ont dû développer des théories très élaborées. Ils sont en effet les pères de la mécanique céleste, la science qui explique le mouvement des corps célestes en interaction gravitationnelle. Copernic a proposé l'héliocentrisme, première grande révolution de son époque. Mais il supposait toujours que les planètes décrivaient des orbites circulaires autour du Soleil. Kepler nous a fait part de ses trois lois, dont la première nous apprend que le mouvement des planètes est en réalité elliptique et que le Soleil se situe à l'un des foyers. Galilée a permis de confirmer ces faits grâce à ses observations. Cependant, Kepler ne pouvant expliquer le pourquoi du mouvement des astres, nous considérons actuellement que la plus grande avancée fut la découverte de la gravitation universelle par Newton. C'est cette découverte qui est à la base de la mécanique céleste, car on y décrit une véritable dynamique des corps célestes et non plus de simples modèles de cinématique.

Au fur et à mesure de l'avancement des théories, de l'amélioration de la précision des instruments et de l'évolution technologique, la mécanique céleste a pris de l'ampleur et à l'heure actuelle il est quasi impossible de développer de nouvelles théories ou d'améliorer celles qui existent déjà sans utiliser des ordinateurs. Ceux-ci sont nécessaires aux calculs mais également pour la théorie elle-même. En effet, certaines de ces théories font appel à des équations différentielles qui nécessitent l'utilisation d'intégrateurs numériques. D'autres sont basées sur des développements en séries dont le grand nombre de termes nécessite un outil approprié, ce qui explique le développement des manipulateurs symboliques ces dernières années. Ces manipulateurs sont des outils informatiques qui permettent notamment de traiter des séries sous forme de tableaux et d'effectuer des opérations en tout genre sur ces séries.

La théorie de Newton, les formalismes lagrangien et hamiltonien (voir Chapitre 1) et toutes les manipulations qui en découlent font apparaître des systèmes d'équations différentielles où un grand nombre de manipulations algébriques entrent en jeu. De nombreux manipulateurs symboliques ont été développés ces dernières années pour des systèmes en tout genre, mais il existe une quantité de problèmes pour lesquels ces manipulateurs ne sont pas adaptés. C'est notamment le cas dans les systèmes où apparaissent des séries de Poisson (cf. Chapitre 1), qui consistent en un développement limité en excentricité et en inclinaison. Cela arrive régulièrement en mécanique céleste, principalement en théorie des perturbations (cf. Chapitre 1 également). Le calcul relatif aux séries de Poisson (somme, multiplication, différentiation, intégration, etc.) est assez aisé et les problèmes de simplification sont souvent absents grâce à leur forme très particulière. Cependant, l'obstacle principal vient de la manipulation de très grandes séries générées par la théorie des perturbations (plusieurs milliers de termes) et l'intérêt de manipulateurs relatifs aux séries de Poisson prend alors tout son sens.

Le MSNam, le Manipulateur de Séries NAMurois, que nous traiterons ici est un ensemble de sous-routines écrites en FORTRAN 90 qui permettent de manipuler de telles séries de Poisson, comme nous le verrons dans le Chapitre 2. Il a tout d'abord été conçu en FORTRAN 77 par Michèle Moons, puis amélioré et adapté au FORTRAN 90 par Jacques Henrard en 2004.
Cet outil a été beaucoup utilisé pendant plus de vingt-cinq ans par les chercheurs de l'unité de Systèmes Dynamiques des Facultés Universitaires Notre-Dame de la Paix à Namur. Les sujets de recherche ont été très différents : la libration de la Lune (M. Moons), les satellites Galiléens (J. Henrard), les satellites artificiels lunaires (B. De Saedeleer), les satellites géostationnaires (S. Valk), le mouvement des exoplanètes (A.-S. Libert) ou la rotation de Mercure (J. Dufey).
Cependant, l'utilisation et la compréhension des sous-routines sont loin d'être aisées étant donné que le MSNam a pour seul support un petit descriptif des sous-routines, et un nombre très restreint de commentaires dans le code (qui fait près de 3000 lignes !). Pour qu'un nouvel utilisateur puisse se servir du MSNam, il lui faut comprendre la quasi totalité du code, ce qui demande donc un énorme travail de recherche avant de pouvoir utiliser cet outil.

Le but de ce mémoire sera de faire cette recherche une fois pour toutes, afin de rendre le MSNam beaucoup plus facile à utiliser. Il s'agira d'une part d'étoffer le descriptif pour en faire une sorte de "mode d'emploi du MSNam" et de commenter le code lui-même afin de ne pas s'y perdre et toujours devoir se référer au mode d'emploi.
D'autre part, il faudra recoder certaines parties qui sont un mélange de FORTRAN 77 et de FORTRAN 90, harmoniser les routines et rendre certains procédés plus automatiques. Bref, le rendre plus "user-friendly". Il s'agit donc d'un travail principalement de programmation mais qui vise également la compréhension d'éléments de mécanique céleste, qui sont la base même de l'utilisation d'un tel programme.

Le premier chapitre sera consacré aux séries de Poisson en mécanique céleste. Nous y ferons quelques rappels de mécanique céleste et de dynamique, introduirons le problème des n-corps et, à titre d'exemple, développerons la théorie des perturbations dans le cas particulier du problème des trois corps. Nous remarquerons ainsi où et sous quelle forme apparaissent les séries de Poisson dans ce domaine, afin de bien comprendre l'intérêt du développement des manipulateurs de séries.

Ensuite, au Chapitre 2, nous donnerons un mode d'emploi explicite du MSNam. Il s'agira d'une part d'expliquer comment fonctionne le manipulateur, c'est-à-dire savoir comment sont stockées les séries, comment initialiser le programme et comment l'utiliser. D'autre part, chaque sous-routine sera expliquée en détail, afin de comprendre en profondeur le fonctionnement et les subtilités d'un tel manipulateur.

Le Chapitre 3 sera consacré à ce que l'on appelle les opérations vectorielles. Nous y expliquerons la différence entre celles-ci et les « opérations normales » ainsi que leurs intérêts et subtilités.

Le quatrième chapitre décrira toutes les améliorations apportées au MSNam lors de ce travail. Ce chapitre se divisera en deux parties distinctes : les modifications apportées à l'ancien code et l'ajout de nouvelles sous-routines. Dans la première partie, nous nous intéresserons à la manière de rendre le programme plus facile à lire via l'ajout de commentaires et d'en-têtes ainsi que grâce à l'uniformisation du code. Nous trouverons dans le seconde partie le détail complet du fonctionnement des nouvelles sous-routines.

Dans le Chapitre 5, nous nous intéresserons à un exemple concret, basé sur des données réelles du système Uppsilon-Andromedae. Nous verrons notamment comment décrire l'évolution de l'excentricité des planètes de ce système avec le temps.

Le sixième chapitre sera quelque peu particulier. Le but de celui-ci sera d'expliquer le fonctionnement d'une sous-routine du MSNam, `dev2B_fr`. Cette dernière permettant de faire des développements en série de Lie, il

nous faudra commencer par expliquer en quoi consistent les transformées de Lie. Ensuite, nous effectuerons quelques développements à la main, que nous vérifierons et prolongerons à l'aide du MSNam. Enfin, nous décrirons le fonctionnement de la sous-routine.

Le dernier chapitre sera consacré à la comparaison entre le MSNam et un autre manipulateur que nous avons pu découvrir, Chronos. Tout au long du chapitre, nous expliquerons les différences entre les manipulateurs, en donnant ainsi leurs avantages et inconvénients.

Chapitre 1

Les séries de Poisson en mécanique céleste

Avant de se plonger au cœur du MSNam et de la programmation, il est pertinent de voir dans quel domaine et sous quelle forme apparaissent les séries de Poisson.

Nous commencerons donc par quelques rappels de mécanique classique. Nous aborderons dans la Section 1.1 les équations du mouvement de Newton, ainsi que le passage aux formalismes lagrangien et hamiltonien. Nous rappellerons également les avantages du formalisme hamiltonien par rapport au lagrangien.

La Section 1.2 sera consacrée à la mécanique céleste qui, rappelons-le, est une branche des mathématiques et de la physique qui tente d'expliquer le mouvement des corps célestes. Ce mouvement est principalement expliqué par la gravitation, c'est pourquoi nous y introduirons le problème des deux corps et développerons plus en détails le problème des trois corps. Nous y découvrirons comment décrire l'hamiltonien du système en fonction de différents repères de coordonnées.

Ce problème peut être étudié au moyen d'intégrations numériques ou au moyen de théories analytiques. L'une de ces théories analytiques est la théorie des perturbations, que nous développerons à la Section 1.3 en suivant l'article de [Laskar, 1988]. Nous y ferons le développement du terme en $\frac{1}{\Delta}$ qui apparait dans la section précédente. Nous verrons également comment le transformer en série de Poisson.

Nous clôturerons ce chapitre en montrant que les séries de Poisson qui apparaissent en mécanique céleste ont une forme particulière qui explique pourquoi le MSNam a été développé.

1.1 De Newton à Hamilton

Dans cette section nous rappellerons les différents formalismes qui permettent de décrire le mouvement des corps.

La loi de la gravitation universelle découverte par Newton nous donne la valeur de la force appliquée sur deux corps A et B de masses respectives m_A et m_B séparées par une distance r et est donnée par la formule

$$\vec{F}_g = -G\frac{m_A m_B}{r^2}\vec{u}_r,$$

où $G = 6,67 \times 10^{-11}\ m^3 kg^{-1} s^{-1}$ est la constante de gravitation universelle, $\vec{u}_r$ un vecteur unitaire pointant de A vers B et où le signe "$-$" indique que la force est attractive, dans le sens opposé au vecteur $\vec{u}_r$.

La mécanique lagrangienne est utilisée à la place de la mécanique classique de Newton par exemple dans le cas où le système présente des forces de liaison (c'est-à-dire des forces dont l'effet est connu mais dont la forme est inconnue). Dans ce formalisme, les équations du mouvement d'un système à N degrés de liberté dépendent des coordonnées généralisées $\{q_i\}_{i=1,\dots,N}$ et des vitesses correspondantes $\{\dot{q}_i\}_{i=1,\dots,N}$, où $\dot{q}_i = \frac{dq_i}{dt}$. Le lagrangien du système est souvent défini comme la différence de l'énergie cinétique et potentielle

$$\mathcal{L}(q_i, \dot{q}_i, t) = T - V$$

Les équations de Lagrange sont données par

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_i}\right) - \frac{\partial \mathcal{L}}{\partial q_i} = 0 \quad \text{pour } i = 1, \dots, N. \tag{1.1}$$

Comme les équations de Newton, il s'agit d'un système de N équations différentielles du second ordre. Ce formalisme est un passage obligé vers la mécanique hamiltonienne qui, elle, permettra une représentation plus aisée du problème.

En mécanique hamiltonienne, les vitesses généralisées $\dot{q}_i$ sont remplacées par la quantité de mouvement associée, encore appelée moment conjugué

$$p_i = \frac{\partial \mathcal{L}}{\partial \dot{q}_i}.$$

L'hamiltonien du système est défini comme la transformation de Legendre du lagrangien

$$\mathcal{H}(q_i, p_i, t) = \sum_{k=1}^{N} p_k \cdot \dot{q}_k - \mathcal{L}(q_i, \dot{q}_i, t) \tag{1.2}$$

et les équations hamiltoniennes sont données par

$$\dot{p}_i = -\frac{\partial \mathcal{H}}{\partial q_i} \quad ; \quad \dot{q}_i = \frac{\partial \mathcal{H}}{\partial p_i} \quad \text{pour } i = 1, \dots, N. \tag{1.3}$$

Les équations de Hamilton consistent en un ensemble de $2N$ équations différentielles du premier ordre et présentent deux avantages. Tout d'abord, la géométrie associée aux équations hamiltonienne est souvent plus simple que celle des équations de Lagrange et la recherche d'invariants y est plus aisée. De plus, les équations hamiltoniennes sont invariantes sous transformation canonique. A l'inverse des équations de Lagrange où tout changement de coordonnées implique de tout recalculer, les transformations canoniques préservent la structure de phase des équations d'Hamilton. Le calcul préalable est certes plus long (à partir des coordonnées généralisées et du lagrangien, il faut calculer l'hamiltonien, exprimer les vitesses généralisées en fonction des moments conjugués et remplacer celles-ci dans la définition de l'hamiltonien), mais les avantages qui en découlent en valent la peine.

1.2 Mécanique céleste et problème des trois corps

Le problème de base de la mécanique céleste est le problème des deux corps. Il s'agit de décrire le mouvement de deux particules isolées, de masses m_1 et m_2 soumises à une force gravitationnelle de type action-réaction. Ce problème est abondamment abordé dans les ouvrages (notamment [Murray et Dermott, 1999]) et ne sera pas traité ici. Cependant, ne considérer que deux corps est une approximation généralement trop imprécise dans les faits. En effet, il faut considérer les perturbations dues aux autres corps célestes qui entourent les deux corps.
Le problème planétaire des trois corps est un bon exemple de ces perturbations. On y considère un des corps (l'étoile) de masse bien supérieure à celles des deux autres corps (les planètes). Le but est alors de modéliser et décrire le mouvements des planètes en interaction avec le corps central, mais sans perdre de vue l'interaction entre les planètes elles-mêmes. Ce problème peut encore se généraliser en problème des n-corps lorsque l'on considère non plus trois, mais n corps en interaction gravitationnelle.
Dans la suite, nous nous restreindrons à l'étude du problème des trois corps, qui suffira amplement à introduire la théorie des perturbations et l'apparition des séries de Poisson. La présente étude est inspirée de [Libert, 2007].

Considérons un corps central (une étoile) de masse m_0 et deux planètes de masses m_1 et m_2 supposées petites par rapport à m_0.
Si l'on se place dans le repère barycentrique, le repère dont l'origine est le centre de masse des trois corps, la seconde loi de Newton ($\sum \vec{F} = m\vec{a}$) nous donne

$$m_i \frac{d^2\vec{u_i}}{dt^2} = -Gm_i \sum_{j\neq i} m_j \frac{\vec{u_i} - \vec{u_j}}{\|\vec{u_i} - \vec{u_j}\|^3} \qquad (i = 0, 1, 2), \tag{1.4}$$

où $\vec{u_i}$ pour $i = 0, 1, 2$ désigne la position de chaque corps i par rapport à l'origine et G est la constante de gravitation de Newton. Nous noterons $\vec{u_i} = (\alpha_i, \beta_i, \gamma_i)$ les coordonnées du vecteur $\vec{u_i}$.
Il s'agit d'un système de neuf équations différentielles du second ordre, que l'on peut détailler sous la forme

$$\left\{ \begin{array}{rcl} m_i \dfrac{d^2\alpha_i}{dt^2} & = & \dfrac{\partial F}{\partial \alpha_i} \\ m_i \dfrac{d^2\beta_i}{dt^2} & = & \dfrac{\partial F}{\partial \beta_i} \\ m_i \dfrac{d^2\gamma_i}{dt^2} & = & \dfrac{\partial F}{\partial \gamma_i} \end{array} \right. \qquad (i = 0, 1, 2) \tag{1.5}$$

où, si l'on note $r_{jk} = \|\vec{u_j} - \vec{u_k}\|$ la distance entre le corps j et le corps k,

$$F = G\left(\frac{m_0 m_1}{r_{01}} + \frac{m_0 m_2}{r_{02}} + \frac{m_1 m_2}{r_{12}}\right). \tag{1.6}$$

Cependant, grâce aux symétries du problème (par rotation et par translation dans l'espace $\mathbb{R}^3$) nous allons pouvoir rechercher des intégrales premières, changer de système de coordonnées et formuler l'hamiltonien du système dans ces nouvelles coordonnées.

L'intégrale première que nous utiliserons d'abord consiste à réaliser la réduction du centre de masse. Cela permettra de ramener le problème à un système de six équations du second ordre. Nous allons pour cela exprimer l'hamiltonien correspondant au système 1.4 dans les coordonnées de Jacobi. Le principe des coordonnées de Jacobi est de repérer chaque corps P_i par rapport au barycentre (G_{i-1} à la Figure 1.1) des i corps précédents $P_0, \ldots, P_{i-1}$. Définissons $\vec{r_i} = (x_i, y_i, z_i)$ le vecteur position du corps de masse m_i dans le repère jacobien.
Si l'on note comme précédemment $(\alpha_0, \beta_0, \gamma_0, \alpha_1, \beta_1, \gamma_1, \alpha_2, \beta_2, \gamma_2)$ les coordonnées respectives du corps central et de deux planètes dans un repère barycentrique, les coordonnées de Jacobi pour la première composante s'écrivent :

$$\begin{aligned} x_1 &= \alpha_1 - \alpha_0 \\ x_2 &= \alpha_2 - \alpha_0 - \kappa_1(\alpha_1 - \alpha_0), \end{aligned} \tag{1.7}$$

où $\kappa_1 = \dfrac{m_1}{m_0 + m_1}$.
On peut faire de même pour les composantes y_i et z_i.

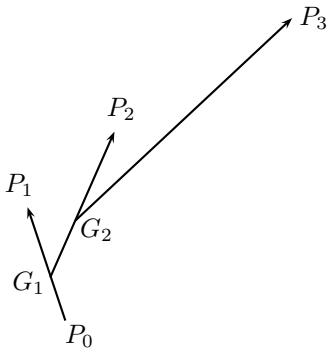


FIGURE 1.1 – Coordonnées de Jacobi

Une série de calculs permet d'obtenir les équations différentielles suivantes en x :

$$\frac{d^2x_1}{dt^2} = \frac{m_0 + m_1}{m_0 m_1}\frac{\partial F}{\partial x_1} \tag{1.8}$$

$$\frac{d^2x_2}{dt^2} = \frac{m_0 + m_1 + m_2}{(m_0 + m_1)m_2}\frac{\partial F}{\partial x_2}. \tag{1.9}$$

Les équations en y et en z ont la même forme que les précédentes.

Afin d'améliorer la compréhension de la géométrie du problème, les mathématiciens et astronomes expriment les coordonnées des astres dans des variables qui correspondent aux éléments orbitaux. Ceux-ci sont représentés aux figures 1.2 et 1.3 et ont la particularité d'être adaptés au problème car ils permettent aisément de repérer la position d'un corps céleste.
La figure 1.2 représente le plan orbital d'un corps céleste dans l'espace. On y introduit i l'inclinaison de l'orbite par rapport au plan de référence, Ω la longitude du nœud ascendant et ω l'argument du péricentre qui est l'angle dans le plan orbital entre la ligne des nœuds et la direction du péricentre.

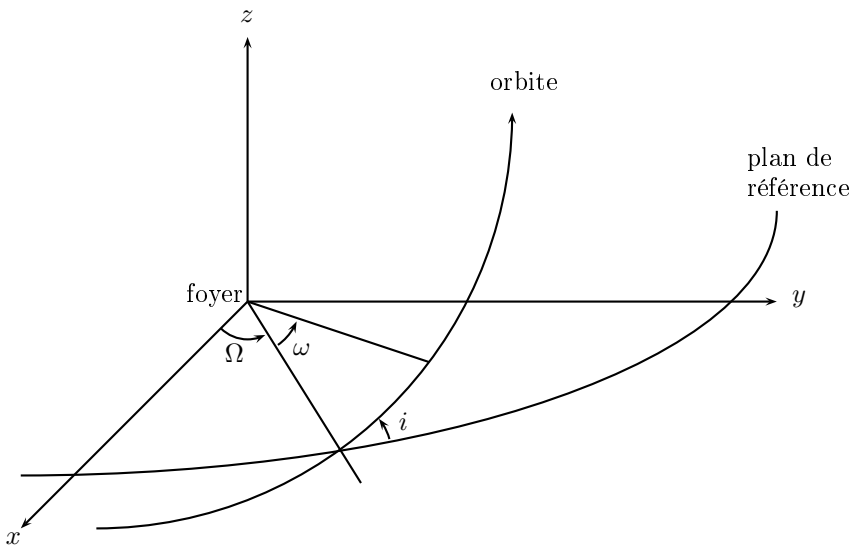


FIGURE 1.2 – Configuration du plan orbital

La figure 1.3 représente la configuration géométrique classique d'une ellipse de demi-grand axe a et d'excentricité e. On y introduit la position du corps céleste sur l'ellipse via l'anomalie vraie f (l'angle entre la direction du péricentre et la position courante d'un objet sur son orbite) ou via l'anomalie excentrique E (l'angle entre la direction du péricentre et la position courante d'un objet sur son orbite, projetée sur le cercle exinscrit perpendiculairement au grand axe de l'ellipse, mesuré au centre de celle-ci).

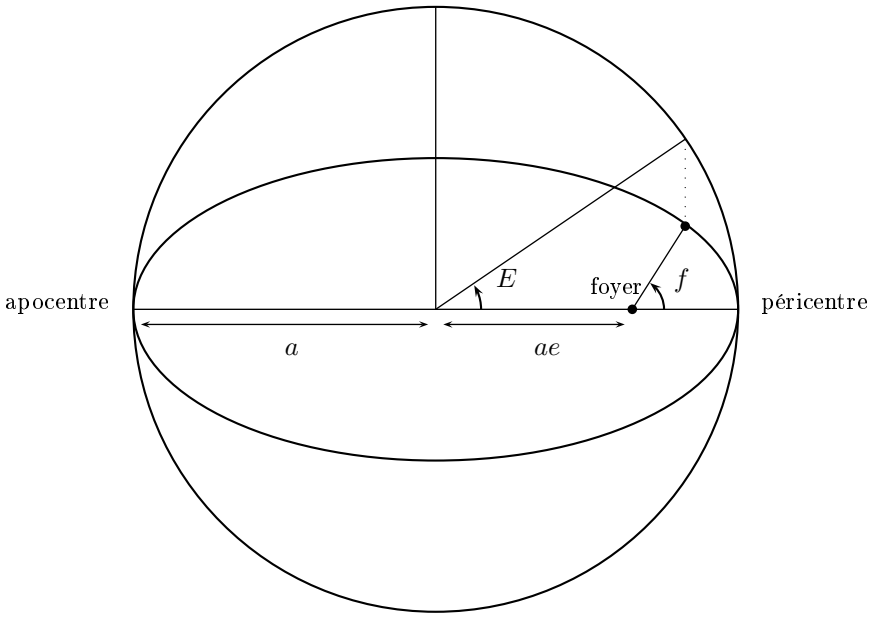


FIGURE 1.3 – Géométrie de l'ellipse et position d'un astre

Les éléments orbitaux $(a, e, i, \omega, \Omega, f)$ ne constituent cependant pas un repère de coordonnées canoniques, c'est pourquoi on utilise une formulation dérivée des éléments orbitaux, appelés éléments ou variables de Delaunay. Pour ce faire, on définit M l'anomalie moyenne via l'équation de Kepler

$$M = E - e \sin E = n(t - \tau),$$

où $n = \sqrt{\mu/a^3}$ avec μ une constante qui dépend des masses et de la constante de gravitation G, et où l'on a introduit τ le temps de passage au péricentre. On obtient alors les variables de Delaunay (l, g, h, L, G, H) où

$l(= M)$	anomalie moyenne	L	$= \sqrt{\mu a}$
$g(= \omega)$	argument du péricentre	G	$= \sqrt{\mu a(1-e^2)}$
$h(= \Omega)$	longitude du nœud ascendant	H	$= \sqrt{\mu a(1-e^2)}\cos i.$

Moyennant l'hypothèse de planètes de petites masses, nous pouvons limiter le calcul de l'hamiltonien aux termes de second degré des masses. L'hamiltonien du problème des trois corps dans les variables canoniques de Delaunay exprimées dans le repère de Jacobi s'écrit alors

$$\mathcal{H} = -\frac{Gm_0m_1}{2a_1} - \frac{Gm_0m_2}{2a_2} - Gm_1m_2\left[\frac{1}{|\vec{r_2} - \vec{r_1}|} - \frac{(\vec{r_1}|\vec{r_2})}{r_2^3}\right], \tag{1.10}$$

où a_1 et a_2 sont les demi-grands axes respectifs des ellipses décrites par les planètes de masse m_1 et m_2 autour de l'étoile. Les deux premiers termes de cette expression ne sont rien d'autre que les hamiltoniens de problèmes de deux corps modélisant les interactions entre le corps central m_0 et les planètes m_1 et m_2 respectivement. Le dernier terme représente la perturbation de ces problèmes de deux corps induite par les interactions des planètes entre elles. Ce terme est lui-même composé d'une partie appelée partie directe, qui correspond à l'inverse de la distance entre les deux planètes, et d'une seconde plus complexe appelée partie indirecte.

Le lecteur intéressé par les détails des calculs qui précèdent peut consulter les Annexes B et C de [Libert, 2007].
Remarquons enfin que nous avons obtenu une expression qui ne dépend plus que des positions $\vec{r_1}$ et $\vec{r_2}$, soit un hamiltonien à six degrés de liberté. Les équations d'Hamilton donnent donc lieu à un système de douze équations du premier ordre. Il s'agit donc d'un système identique à celui des équations de Newton ou de Lagrange, mais dont la géométrie permet une meilleure représentation du problème

1.3 Un exemple de série de Poisson : le développement de la fonction perturbatrice

Dans la section précédente, nous avons obtenu l'hamiltonien du problème des trois corps sous la forme décrite en (1.10). La principale difficulté dans le calcul de la fonction perturbatrice se trouve dans le développement de l'inverse de la distance

$$\frac{1}{\Delta} = \frac{1}{|\vec{r_2} - \vec{r_1}|}. \tag{1.11}$$

Pour ce faire, nous considérons une configuration du système telle qu'elle est décrite à la Figure 1.4, où ψ désigne l'angle entre les deux vecteurs positions.

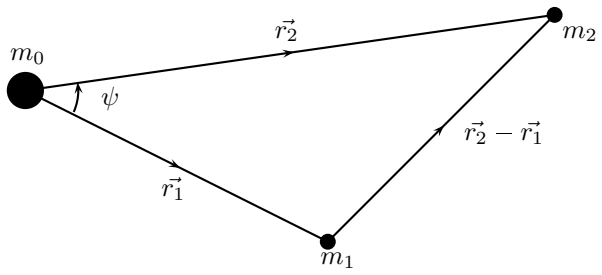


FIGURE 1.4 – Configuration du problème des trois corps

Dans cette configuration, nous considérons les positions des deux planètes par rapport à l'étoile dans les coordonnées de Jacobi. Il est important de remarquer que chacune de ces positions (et par conséquent l'angle ψ) dépend implicitement des éléments orbitaux.

Nous omettrons pour l'instant ce fait, et allons développer le terme de perturbation en fonction de ψ uniquement.
Par la règle du cosinus, nous avons immédiatement

$$\Delta^2 = |\vec{r_2} - \vec{r_1}|^2 = r_1^2 + r_2^2 - 2r_1 r_2 \cos\psi \tag{1.12}$$

ou de manière équivalente

$$\frac{1}{\Delta} = \left(r_1^2 + r_2^2 - 2r_1 r_2 \cos\psi\right)^{-\frac{1}{2}}. \tag{1.13}$$

En mettant le terme $\frac{1}{r_2}$ en évidence et en définissant $\rho = \dfrac{r_1}{r_2}$ (on suppose généralement $r_1 < r_2$), l'équation (1.13) devient

$$\begin{aligned} \frac{1}{\Delta} &= \frac{1}{r_2}\left[\left(1 - 2\frac{r_1}{r_2}\cos\psi + \frac{r_1}{r_2}\right)^2\right]^{-\frac{1}{2}} \\ &= \frac{1}{r_2}\left(1 + \rho^2 - 2\rho\cos\psi\right)^{-\frac{1}{2}}. \end{aligned} \tag{1.14}$$

Lorsque ρ est petit, on effectue un développement en série par rapport à ρ et on obtient immédiatement

$$\frac{1}{\Delta} = \frac{1}{r_2}\sum_{i=0}^{\infty}\rho^i P_i(\cos\psi), \tag{1.15}$$

où les P_i sont des polynômes de Legendre et vérifient

$$\begin{aligned} P_0(\cos\psi) &= 1 \\ P_1(\cos\psi) &= \cos\psi \\ P_2(\cos\psi) &= \frac{1}{2}(3\cos^2\psi - 1) \\ &\vdots \end{aligned}$$

Dans ce cas, on peut déjà voir apparaître un développement en série, que l'on pourrait considérer comme une série de Poisson. En effet, on y reconnaît un coefficient $\frac{1}{r_2}$, un terme polynomial ρ^i et le cosinus d'un angle (qui dépend implicitement des éléments orbitaux, c'est-à-dire d'autres angles).
Cependant, dans le cas planétaire, ρ est rarement petit et on peut plutôt effectuer un autre développement. En effet, puisque $\rho = \frac{r_1}{r_2}$ et que les positions r_1 dépendent implicitement des demi-grands axes a_1 et a_2 et donc de e et de i par définition des ellipses, nous effectuons un développement en fonction des excentricités et des inclinaisons. C'est ce développement qui nous mènera à un exemple concret de série de Poisson.

Remarquons d'abord que les vecteurs positions $\vec{r_1}$ et $\vec{r_2}$ peuvent être exprimés en fonction des éléments orbitaux $(a, e, i, \omega, \Omega, f)$ décrits plus haut selon la formule :

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \cos\Omega & -\sin\Omega & 0 \\ \sin\Omega & \cos\Omega & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos i & -\sin i \\ 0 & \sin i & \cos i \end{pmatrix} \begin{pmatrix} \cos\omega & -\sin\omega & 0 \\ \sin\omega & \cos\omega & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} r\cos f \\ r\sin f \\ 0 \end{pmatrix} \tag{1.16}$$

Cela nous permet d'écrire les relations suivantes pour chaque planète

$$\left\{ \begin{array}{rcl} x_i &=& r_i \left[\cos\Omega_i \cos(\omega_i + f_i) - \sin\Omega_i \sin(\omega_i + f_i)\cos i_i\right] \\ y_i &=& r_i \left[\sin\Omega_i \cos(\omega_i + f_i) - \cos\Omega_i \sin(\omega_i + f_i)\cos i_i\right] \\ z_i &=& r_i \left[\sin(\omega_i + f_i)\sin i_i\right] \end{array} \right. \quad \text{pour } i = 1, 2. \tag{1.17}$$

Nous définissons un nouvel angle

$$\Psi = \cos\psi - \cos(\lambda_1 - \lambda_2), \tag{1.18}$$

où $\lambda_i = f_i + \omega_i + \Omega_i$ est la longitude moyenne.
On peut alors reprendre le développement de (1.13) :

$$\frac{1}{\Delta} = \left(r_1^2 + r_2^2 - 2r_1 r_2 \cos\psi\right)^{-\frac{1}{2}} \tag{1.19}$$

$$= \left[r_1^2 + r_2^2 - 2r_1 r_2 \left(\cos(\lambda_1 - \lambda_2) + \Psi\right)\right]^{-\frac{1}{2}}. \tag{1.20}$$

Un développement en série de Taylor permet d'obtenir la relation

$$\frac{1}{\Delta} = D + \sum_{n \geqslant 1} (-1)^n \frac{(2n-1)!!}{(2n)!!} F D^{2n+1}, \tag{1.21}$$

où $D = \left[a_2^2 + a_1^2 - 2a_1 a_2 \cos(\lambda_1 - \lambda_2)\right]^{-\frac{1}{2}}$, F est une fonction des éléments de Delaunay et où l'on introduit la double factorielle définie par

$$n!! = n(n-2)(n-4)\ldots$$

On peut ensuite développer le terme D^{2n+1} en série de Fourier afin d'obtenir

$$D^{2n+1} = \sum_{k \geqslant 1} B_k^{(2n+1)/2} \cos k(\lambda_1 - \lambda_2), \tag{1.22}$$

où les coefficients $B_k^{(2n+1)/2}$ sont appelés coefficients de Laplace et sont des fonctions hypergéométriques dépendant uniquement du rapport des demi-grands axes.

Dans le cadre de ce mémoire (voir [Libert, 2007] ou [Murray et Dermott, 1999] pour plus de détails), nous admettrons que tous les termes des séries présentes ci-dessus peuvent être rassemblés sous une forme compacte afin d'obtenir une fonction hamiltonienne de la forme

$$\begin{aligned}\mathcal{H} &= -\frac{Gm_0m_1}{2a_1} - \frac{Gm_0m_2}{2a_2} \\ &\quad -\frac{Gm_1m_2}{a_2} \sum_{k,i_l,j_l,l\in\underline{4}} A_{i_l}^{k,j_l} e_1^{|j_1|+2i_1} e_2^{|j_2|+2i_2} \left(\sin\frac{i_1}{2}\right)^{|j_3|+2i_3} \left(\sin\frac{i_2}{2}\right)^{|j_4|+2i_4} \cos\Phi, \end{aligned} \tag{1.23}$$

où $\Phi = (k + j_1 + j_3)\lambda_1 - (k + j_2 + j_4)\lambda_2 - j_1\varpi_1 + j_2\varpi_2 - j_3\Omega_1 + j_4\Omega_2$ est la combinaison d'angles. Les indices $(k, i_l, j_l, l \in \underline{4})$ sont des entiers et les coefficients $A_{i_l}^{k,j_l}$ sont réels et dépendent du rapport des demi-grands axes.

La série qui apparaît dans (1.23) est ce que l'on appelle une série de Poisson, c'est-à-dire une série de Fourier avec λ_i, ω_i et Ω_i comme variables angulaires et dont les coefficients sont polynomiaux en les variables e_i et $sin(i_i/2)$.

Le MSNam a été créé dans le but de manipuler de telles séries, plus particulièrement sous la forme :

$$S = \sum_{\substack{i_1,\cdots,i_p \\ j_1,\cdots,j_q}} A \begin{smallmatrix} i_1,\cdots,i_p \\ j_1,\cdots,j_q \end{smallmatrix} x_1^{i_1}\cdots x_p^{i_p} \left\{ \begin{array}{c} \sin \\ \cos \end{array} \right\} (j_1\phi_1 + \cdots + j_q\phi_q) \tag{1.24}$$

c'est-à-dire des sommes de Fourier en les q angles $\phi_1, \cdots, \phi_q$ dont les coefficients sont polynomiaux en les p variables $x_1, \cdots, x_p$. Les *arguments* $(j_1, \cdots, j_q)$ et les *exposants* $(i_1, \cdots, i_p)$ sont des nombres entiers et les *coefficients* $A_{\{\cdots\}}$ sont des réels.

En partant d'un problème simple de mécanique céleste, nous avons vu comment transformer l'hamiltonien du problème des trois corps sous forme de série de Poisson. Le développement peut être généralisé au problème des n-corps. Nous aurions également pu choisir de développer les polynômes de Legendre de (1.15) sous forme de série de Poisson. De plus, bien d'autres problèmes de mécanique céleste font appel à de telles séries.

Chapitre 2

Mode d'emploi du MSNam

Ce chapitre a pour but de constituer un guide complet du MSNam en expliquant d'une part son fonctionnement et d'autre part en décrivant ses sous-routines. La description des sous-routines est basée sur les notes de Jacques Henrard [Aide MSNam] (voir Annexe A). Il explique dans cet article les paramètres d'entrée et de sortie des sous-routines et décrit brièvement en quoi consiste chaque routine. En se basant sur ce modèle, nous proposons ici à l'utilisateur un guide plus complet, qui décrit non seulement le fonctionnement de chaque sous-routine, mais également comment utiliser le MSNam.

Afin de fixer les notations, rappelons que le MSNam est un ensemble de sous-routines écrites en FORTRAN 90 qui permettent de manipuler des séries de Poisson de la forme :

$$S = \sum_{\substack{i_1,\cdots,i_p \\ j_1,\cdots,j_q}} A_{\substack{i_1,\cdots,i_p \\ j_1,\cdots,j_q}} \; x_1^{i_1}\cdots x_p^{i_p} \left\{ \begin{array}{c} \sin \\ \cos \end{array} \right\} (j_1\phi_1 + \cdots + j_q\phi_q) \tag{2.1}$$

c'est-à-dire des sommes de Fourier en les q angles $\phi_1, \cdots, \phi_q$ dont les coefficients sont polynomiaux en les p variables $x_1, \cdots, x_p$.
Les *arguments* $(j_1, \cdots, j_q)$ et les *exposants* $(i_1, \cdots, i_p)$ sont des nombres entiers et les *coefficients* $A_{\{\ldots\}}$ sont des réels en double précision.

Après quelques informations sur les manipulateurs symboliques, nous passerons en revue dans ce chapitre les paramètres utilisés par le MSNam avant de décrire où et comment sont stockées les séries. Ensuite, nous expliquerons comment initialiser le manipulateur et terminerons par le descriptif des sous-routines, classées en fonction de leur rôle.

2.1 Sur les manipulateurs symboliques

Les algorithmes qui peuvent être implémentés afin d'effectuer des manipulations sur des séries ainsi que leur efficacité dépendent de la manière dont la série a été codée. Voyons quelques possibilités de codage et notons-en les avantages et inconvénients. Par souci de notation, nous ne considérerons que des polynômes en quelques variables, la généralisation aux séries de Poisson étant triviale.

Considérons un polynôme en deux variables

$$P = X^2 + XY + X^2Y + XY^2 + Y^3$$

Un premier schéma de codage, que l'on pourrait qualifier de « complet », consiste à coder

a) le nom des variables

b) l'exposant

c) le coefficient numérique

Chaque terme est une structure (un « record ») dépendant de la taille des variables et la série est une liste de telles structures

$P \rightarrow$

X	2	$1.\cdots D0$		
X	1	Y	1	$1.\cdots D0$
⋮				
Y	3	$1.\ldots D0$		

Un deuxième schéma, que l'on pourrait qualifier de « codifié », consiste à supposer que l'on a un polynôme en les variables X et Y et que les noms des variables n'ont pas besoin d'être répétés, cette supposition pouvant être vraie pour toute l'application ou seulement pour la série en cours (dans ce cas le nom des variables doit apparaître quelque part en en-tête de la série).

$P \rightarrow$

les variables sont X, Y.		
(X)	(Y)	(coef)
2	0	$1.\cdots D0$
1	1	$1.\cdots D0$
⋮		
0	3	$1.\cdots D0$

Un troisième schéma, que l'on pourrait qualifier de « tableau », consiste en la création d'une correspondance biunivoque entre les termes possibles de la série.

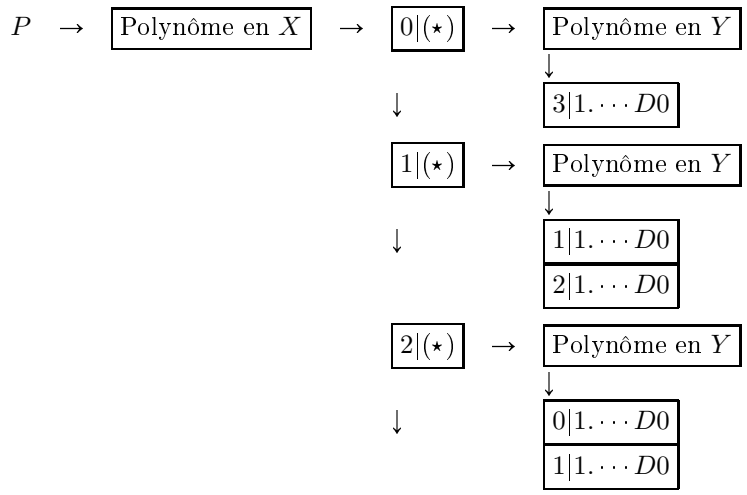
$P \rightarrow$ Tableau à double entrée indexé par les exposants de X et Y dont le maximum est 3

	(X^0)	(X^1)	(X^2)	(X^3)
(Y^0)	0.	0.	1.	0.
(Y^1)	0.	1.	1.	0.
(Y^2)	0.	0.	0.	0.
(Y^3)	1.	0.	0.	0.

Remarquons que de schéma en schéma, moins d'informations sont codées et de plus en plus sont implicites. Dans le dernier, par exemple, seuls les coefficients sont codés, le nom des variables et les exposants étant déduits implicitements. Ce caractère implicite mène souvent à des algorithmes plus efficaces mais moins flexibles.

Chaque méthode à ses avantages et inconvénients. Dans le cas d'un polynôme en un grand nombre de variables mais avec peu de termes, le premier schéma est plus efficace. Dans le cas contraire, c'est le troisième qui s'avère le plus intéressant. La plupart des programmes manipulant des séries de Poisson ont adopté un schéma proche du second qui est, en quelque sorte, un compromis entre les deux.

Il existe cependant une autre méthode pour stocker un polynôme en n variables. Nous pouvons le considérer comme un polynôme à p variables (où $p \leqslant n$) dont les coefficients sont des polynômes à $n - p$ variables. Dans cette représentation récursive, notre exemple s'écrirait comme



Bien que cette représentation peut sembler quelque peu compliquée, elle s'avère en fait être très productive. Nous allons à présent considérer deux algorithmes de base qui peuvent faire la différence dans le cas de problèmes de grande taille.

Le premier concerne le produit de séries tronquées. Nous manipulons souvent des séries de Poisson tronquées, dont le nombre de terme croît rapidement avec la taille des coefficients. Considérons un exemple typique d'une série de Poisson contenant 10^n termes, dont le coefficient est de l'ordre de 10^{-n}, et que l'on a tronqué à 10^{-4}. Elle contient 1 terme d'ordre 1, 10 termes d'ordre 0.1 et jusqu'à 10000 termes d'ordre 10^{-4}. Afin de multiplier deux séries de ce type, il faut effectuer plus de 10^8 produits élémentaires.

Mais la série résultante ne sera certainement pas précise jusqu'à 10^{-8}, qui sera cependant l'ordre de la plupart des produits élémentaires effectués. Nous pouvons complètement éviter d'effectuer une partie des produits si la série à été au préalable triée selon la taille des coefficients. De ce fait, si l'on fixe la précision à 10^{-5} afin d'autoriser une certaine accumulation, nous n'effectuons cependant que 4×10^4 produits élémentaires, ce qui est un gain considérable.

Le second algorithme concerne la recherche d'un terme particulier dans une série. Cette fonction crutiale se retrouve au sein de quasiment toutes les autres. Par exemple, dans un produit $A \cdot B \to C$, chaque produit élémentaire d'un terme de A avec un terme de B est suivit (dans la plupart des algorithmes) par une recherche sur C afin de vérifier si un terme du même type existe déjà. Dans une recherche séquentielle, le nombre de comparaisons peut être aussi élevé que le nombre de termes déjà présents dans C (disons N), alors que pour une recherche dichotomique, le nombre maximum de comparaisons est réduit à $\log_2 N$. Dans le cas $N = 10000$, cela représente une réduction d'un facteur 1000. Une telle recherche dichotomique peut être implémentée via un algorithme d'arbre binaire équilibré (*balanced tree algorithm* en anglais), comme proposé par [Knuth, 1973] si l'on utilise l'ordre lexicographique des variables.

Pour ces deux raisons, le stockage des séries par le MSNam a été codé de manière à respecter ces règles. Elles sont principalement observables dans les sous-routines `PRODUCT, ORDER_SIZE` et `ORDER_CODE`.

2.2 Le module `parameters`

Avant de voir comment utiliser le MSNam et comment sont stockées les séries, il est important de présenter les paramètres principaux du MSNam. Le module `parameters` contient les paramètres qui doivent être définis

par l'utilisateur. Il s'agit de la seule partie du MSNam qui peut (et qui doit) être modifiée par l'utilisateur avant l'utilisation du manipulateur.
Les paramètres peuvent être classés en deux groupes : ceux qui doivent être spécifiés avant l'utilisation et ceux qui peuvent être modifiés mais qui sont généralement fixés.
Remarque : dans le code, les paramètres sont précédés des lettres « `MS` » , nous les omettrons dans ce travail afin d'alléger l'écriture.

Les paramètres à spécifier absolument

- ▷ `nvt` : Nombre de variables trigonométriques, c'est-à-dire le nombre q apparaissant dans la série.

- ▷ `nvp` : Nombre de variables polynomiales, c'est-à-dire le nombre p apparaissant dans la série.

- ▷ `nvar` : Nombre total de variables, automatiquement assigné comme `nvt` + `nvp`.

- ▷ `namevt` : Vecteur de dimension `nvt` qui contient le nom donné aux variables trigonométriques, toutes codées sous forme de chaîne de caractères de longueur 4.
  Ces noms sont utilisés principalement lors de l'affichage des séries, mais aussi dans certaines sous-routines afin d'identifier avec quelle variable (relative à une action) nous sommes en train de travailler (pour des dérivées partielles par exemple). Ces noms doivent être modifiés pour correspondre au problème traité par l'utilisateur. Par exemple, dans le cadre du problème des trois corps, le nom des variables trigonométriques correspondant aux inclinaisons des deux petits corps pourraient être `' i1 '` et `' i2 '` (où nous avons ajouté un espace au début et à la fin afin d'avoir 4 caractères.

- ▷ `namevp` : Vecteur de dimension `nvp` qui contient le nom donné aux variables polynomiales, toutes codées sous forme de chaîne de caractères de longueur 4.
  Ces noms sont utilisés principalement lors de l'affichage des séries, mais aussi dans certaines sous-routines afin d'identifier avec quelle variable (relative à une action) nous sommes en train de travailler (dérivées, produits, mise à l'échelle). Ces noms doivent être modifiés pour correspondre au problème traité par l'utilisateur.

Les paramètres modifiables si nécessaire

- ▷ `accuracy` : A la fin de certaines sous-routines, à savoir `ACUM, PROD, SCALE, SCALEP`, les termes de la série dont la valeur absolue du coefficient est plus petite que `accuracy` sont supprimés afin d'éliminer les termes négligeables. Il est conseillé de fixer `accuracy` à 10^{-14}.

- ▷ `nwords` : Nombre de mots autorisé pour coder les arguments et les exposants de chaque terme d'une série. Comme nous le verrons à la Section 2.7, les arguments et exposants sont stockés sous forme codée. Ce code est un nombre entier pour lequel le MSNam réserve un certain nombre de places (par exemple 328 prend trois places). Ce nombre de places est appelé nombre de mots en programmation. Il dépend du nombre d'arguments et d'exposants ainsi que des valeurs maximales de ceux-ci. `nwords` est généralement fixé à 5 afin de coder des entiers de 32 bits (puisque $32 = 2^5$).

- ▷ `storelength` : Nombre maximum de termes présents dans toutes les séries à un moment donné. Le MSNam va réserver dans le module `TABLES` un tableau `TABLE(nwords,storelength)` afin de stocker les arguments, exposants ainsi que l'indicateur qui représente le sinus ou le cosinus. `storelength` est généralement fixé à 600000 mais peut être modifié pour traiter des problèmes plus gourmands en stockage.

- ▷ `nmaxser` : Nombre maximum de séries présentes en même temps. Il est généralement fixé à 200 mais peut aussi être modifié dans le cadre de problèmes très grands.

▷ `ordmax` : Ordre maximum des sous-routines vectorielles. Il s'agit en fait du nombre maximum d'opérations consécutives que feront les sous-routines vectorielles. Nous verrons au Chapitre 3 que la complexité des opérations croît exponentiellement avec l'ordre. On limite donc généralement `ordmax` à 20.

▷ `nmax` : Vecteur de dimension `nvar` qui indique pour chaque argument et exposant, la valeur maximale que peut prendre sa valeur absolue.
Si $2^{n-1} \leqslant$ `nmax(k)` $< 2^n$, le code pour la variable `k` prendra n bits. Puisque le nombre de mots est généralement fixé à 5 (voir `nwords` ci-dessus), la valeur maximale `nmax` est fixée à $2^n - 1$, c'est-à-dire dans notre cas à 31.
Nous pouvons résumer ce résultat dans un tableau de la manière suivante :

$$\begin{array}{lcl} \text{nmax} = 3 & \rightarrow & 2 \text{ bits} \\ \text{nmax} = 7 & \rightarrow & 3 \text{ bits} \\ \text{nmax} = 15 & \rightarrow & 4 \text{ bits} \\ \text{nmax} = 31 & \rightarrow & 5 \text{ bits} \\ \text{nmax} = 63 & \rightarrow & 6 \text{ bits} \\ & \vdots & \end{array}$$

Dans certains cas, l'utilisateur voudra coder des arguments et exposants ayant des valeurs maximales différentes. Cela peut être le cas si certains arguments prennent une faible valeur et d'autres une plus grande. Il y a alors moyen de récupérer la place inutilisée par les arguments de faible valeur pour la donner à ceux en ayant une plus importante. Ce genre de manipulation est réservée à un utilisateur expérimenté.

2.3 Avant de commencer à programmer

Comme nous venons de le voir, avant de commencer à programmer avec le MSNam, il faut modifier certains paramètres du module `MSparameters` dans le fichier `MSNam.f90`.
Une erreur fréquente est de modifier `nvt` et `nvp` ainsi que le nom des variables `namevt` et `namevp` puis d'oublier de modifier la quantité de « 31 » qui apparaissent dans `nmax`.
Nous rappelons qu'il est impératif que les noms de variables soient des caractères de longueur 4 et qu'il faut donc les raccourcir ou ajouter des espaces si nécessaire.
Nous rappelons également qu'il est déconseillé de modifier d'autres paramètres, sauf si le problème le requiert.

2.4 Le module "TABLES"

Ce module utilise les paramètres définis précédemment afin de créer les tableaux qui seront utilisés pour coder les séries, ainsi que certains paramètres nécessaires au bon fonctionnement des sous-routines. Il sera automatiquement utilisé par toutes les sous-routines du MSNam et ne doit pas être modifié par l'utilisateur. Afin d'être complet, citons les paramètres introduits par ce module :

▷ `tabcoef` : un vecteur de réels en double précision, de dimension `storelength` qui contient les coefficients des termes des séries.

▷ `TABLE` : un tableau d'entiers de dimensions (nwords, storelength) qui contient la version codée des exposants des variables polynomiales, des coefficients des variables trigonométriques et de l'indicateur sinus-cosinus. Avec `tabcoef`, ce tableau permet de stocker les termes des séries.

▷ `free` : un entier qui indique la position de la prochaine case « libre ».

- ▷ `nser` : un entier qui indique le nombre de séries actives.
- ▷ `begin` : un vecteur d'entiers de dimension `nmaxser` indiquant la position du premier terme de chaque série dans l'espace de stockage.
- ▷ `lenght` : un vecteur d'entiers de dimension `nmaxser` indiquant la taille (le nombre de terme) de chaque série.
- ▷ `nvw` : un vecteur d'entiers de dimension `nwords`. Il détermine la dispersion du nombre de variables pour le codage des séries. La somme des valeurs du vecteur vaut `nvar`. Si `nvar` $\leqslant$ `nwords`, seule premier élément est initialisé. Si `nwords`$\leqslant$`nvar`$\leqslant$`2*nwords`, les deux premiers éléments sont initialisés, et ainsi de suite.
- ▷ `shift` : un vecteur d'entiers de dimension `nvar` qui détermine le décalage à effectuer lors du codage des séries.
- ▷ D'autres paramètres qui permettent d'afficher des messages formatés en tout genre

Tous ces paramètres sont déclarés dans ce module, et sont initialisés lors de l'initialisation du MSNam avec `START_MS` (voir section suivante).

2.5 Initialisation du MSNam

`START_MS` est la sous-routine qui initialise les quantités nécessaires au fonctionnement des autres sous-routines. Elle doit être appelée avant toute autre sous-routine.
En fait, rappelons que le MSNam n'est pas un programme en soi mais un ensemble de sous-routines. Lors de son utilisation, il faut donc créer son propre code en FORTRAN 90, faire appel au module `TABLES` (l'appel au module `parameters` se faisant automatiquement avec celui-ci) et lancer `START_MS`.
Nous allons ici également introduire quelques variables qui seront utiles pour la section suivante. D'une part nous introduisons la variable `serA` que nous utiliserons comme série et que nous initialisons immédiatement à 0. D'autre part, nous déclarons également les variables `sc,arg,exp` et `coef` qui seront nécessaires pour le stockage des séries.
L'utilisateur peut ensuite déclarer à sa guise toutes les variables qui lui seront nécessaires.

NB : n'oubliez pas de déclarer les variables qui serviront à faire appel aux fonctions entières comme `NBTERM, INDEXPOL, INDEXTRIG` si vous en avez besoin.

Nous pouvons imaginer un programme de la forme suivante :

```
program test
use MStables
implicit none
   integer:: serA=0    ! Parfois A(0:order)=0, voir Chapitre 3
   integer:: sc, arg(1:MSnvt)=0, exp(1:MSnvp)=0
   double precision:: coef
   integer:: NBTERM, INDEXPOL, INDEXTRIG    ! si nécessaire
  ! Déclaration d'autres variables

   call START_MS

  ! Corps du programme
```

```
  ! Appel à certaines sous-routines du MSNam
end program test
```

En pratique, `START_MS` vérifie d'abord la validité des paramètres introduits dans le module `parameters`. Ensuite, elle crée les formats qui permettront d'écrire et de lire les séries. Elle initialise également tous les pointeurs utilisés et finit par créer les emplacements nécessaires aux variables codées sous forme de mots. Dans cette dernière partie, la sous-routine vérifie que l'utilisateur a prévu suffisamment de place pour coder les arguments et exposants.

2.6 Comment sont stockées les séries ?

Chaque série est identifiée par un entier appelé identifiant qui est utilisé dans les appels aux sous-routines. Généralement, il est noté `A, B,` etc. Par abus de langage, nous parlerons souvent de la série `A` au lieu de la série dont l'identifiant est l'entier `A`.
Au départ, tous les identifiants doivent être initialisés à zéro, cela signigie que la série est vide. Ensuite, l'identifiant de la première série créée devient 1, puis le suivant devient 2, etc. Si l'identifiant est négatif, cela signifie que la série est stockée en fonction de la valeur absolue des coefficients (voir `ORDER_SIZE`).
Cet identifiant est utilisé pour connaître la position du premier terme de la série dans les grands tableaux via la variable `begin`. Par exemple, supposons que la série soit identifiée par 3, alors `begin(3)=150` signifie que le premier terme de la série se situe à la 150$^{\text{e}}$ place dans les tableaux.
Remarquons que si deux identifiants sont égaux, cela signifie que les séries sont égales. Par contre, si nous introduisons deux séries égales dans deux identifiants `A` et `B`, ceux-ci seront différents. Cela est dû au fait que le MSNam ne vérifie pas les termes des séries précédentes lorsque l'on en crée une nouvelle.

Les coefficients $A^{i_1,\cdots,i_p}_{j_1,\cdots,j_q}$ sont stockés sous forme de réels en double précision dans le grand vecteur colonne `tabcoef`.

Les arguments, les exposants et l'indication que l'expression trigonométrique est un cosinus (= 0) ou un sinus (= 1) sont codés et stockés dans le grand tableau d'entiers `TABLE` dont la taille (`storelength`) et la forme (`nwords`) sont définies par l'utilisateur via le module `parameters`. Cependant, il ne suffit pas de stocker ces termes directement dans le tableau, on passe par une représentation codée en binaire ("bit representation" en anglais).
Pour ce faire, les vecteurs des arguments $[j_1,\ldots,j_q]$ et des exposants $[i_1,\ldots,i_p]$ de chaque terme d'une série sont concaténés dans le même ordre et placés dans un seul vecteur d'entiers $[k(1),\ldots,k(q+p)]$. A chaque entier $k(l)$ dans la k-liste a été associé un nombre de bits $n(l)$ par le module `parameters`.
Chaque élément du tableau de mots codés se voit ensuite attribuer une valeur entière qui dépend du nombre de mots `nwords`, du nombre de variables `nvar` et de la valeur des variables. De cette manière, le tableau `TABLE` contient des entiers qui représentent le code des arguments, exposants et de l'indicateur sinus-cosinus. L'avantage de cette représentation est de permettre au programme de pouvoir reconnaître les termes qui sont de la même forme afin de faire des opérations algébriques. De manière très simplifiée, si l'on suppose que l'on a deux variables x et y, le MSNam sait qu'il peut par exemple additionner des termes en x entre eux, mais pas des termes en x et en y.
Remarquons que comme chaque mot de 32 bits permet de coder un nombre entier d'arguments ou d'exposants, il est possible que, dans certains mots, des bits soient libres et puissent être utilisés pour coder d'autres arguments ou exposants ayant une plus grande valeur absolue.

Généralement, il n'est pas nécessaire pour l'utilisateur de comprendre cette représentation car le MSNam a été codé de manière à ce que l'on introduise les termes de manière classique et il transforme lui-même ces termes de manière adéquate. Les sous-routines qui permettent de telles manipulations sont reprises à la Section 2.9 : Sous-routines de base.

2.7 Encodage des séries

Afin de compléter un maximum ce descriptif, nous allons expliquer en détail le fonctionnement de la sous-routine `ENCODE` à l'aide d'un exemple pour que l'utilisateur intéressé puisse comprendre comment se passe le codage.

```
!*********************************************************************!
! subroutine ENCODE(code,sc,arg,exp)                                  !
! The subroutine ENCODE codes in the vector code, the sine-cosine flag !
! sc, the arguments arg and exponents exp                             !
!                                                                     !
!*********************************************************************!
subroutine ENCODE(code,sc,arg,exp)
  use MSTABLES
  implicit none
  integer,intent(in):: sc,arg(MSnvt),exp(MSnvp)
  integer,intent(out):: code(MSnwords)
  integer:: j,k,tablearg(MSnvar),sign,start

  ! concatenate the arguments

  tablearg(1:MSnvt)=arg
  tablearg(MSnvt+1:MSnvar)=exp

  ! Validation
  ! test if the arguments are within their ranges

  do k=1,MSnvar
     if(abs(tablearg(k)).gt.MSnmax(k)) then
        MSmessage='error in ENCODE: argument (1) is equal&
             & to (2) and out of range'
        MSintmess=0
        MSintmess(1)=k
        MSintmess(2)=tablearg(k)
        call ERROR
     endif
  enddo

  ! coding

  start=0
  do k=1,MSnwords
     code(k)=0

     do j=start+1,start+MSnvw(k)
        if(tablearg(j).lt.0) then
           sign=1
        else
           sign=0
        endif
        code(k)=2*code(k)+sign
        code(k)=ishft(code(k),MSshift(j))
```

```
        code(k)=code(k)+abs(tablearg(j))
      enddo
      start=start+MSnvw(k)
  enddo
  code(1)=2*code(1)+sc

end subroutine ENCODE
```

La fonction `ISHFT` présente dans la sous-routine `ENCODE` ci-dessus effectue un décalage binaire. Prenons comme exemple le nombre 15, écrit en binaire sous la forme 1111. Effectuer un décalage positif signifie faire un décalage à gauche. Si nous effectuons un décalage d'une place, 1111 devient 11110 qui est égal à 30 en numérotation décimale. Donc `ISHFT(15,1)=30`.

Prenons un exemple simple pour illustrer le processus de stockage des données. Considérons une série en les variables trigonométriques a, b et en les variables polynomiales x, y. Supposons que nous désirons stocker le terme $y\cos(a)$ que nous pouvons représenter dans un tableau comme suit :

sc	a	b	x	y	coef
0	1	0	0	1	1.0D0

Dans cet exemple, nous ne nous préoccupons pas du coefficient puisque celui-ci est stocké dans le vecteur `tabcoef`.

Puisque notre série est composée de 4 variables, et que nous avons fixé `nwords=5`, les vecteurs `nvw` et `shift` sont respectivement donnés par $(4,0,0,0,0)$ et $(5,5,5,5)$ (suite à leur initialisation avec `START_MS`).

La sous-routine `ENCODE` commence par concaténer les arguments des variables trigonométriques et les exposants des variables polynomiales dans un tableau nommé `tablearg`. Dans notre cas, nous avons `tablearg=(1,0,0,1)`. Après quelques vérifications d'usage, la variable `start` est initialisée à la valeur 0 et une première boucle sur `k=1,nwords` est effectuée. La variable `code(k)` est ensuite initialisée à la valeur 0 également.
Il s'ensuit une nouvelle boucle sur `j=start+1,start+nvw(k)`. Dans notre cas, puisque `nvw(2)=nvw(3)=nvw(4)=0`, cela signifie que cette boucle n'aura lieu que dans le cas où `k=1`. Nous pouvons d'ores et déjà conclure que le code restera 0 pour les 4 dernières composantes.

Concentrons nous donc sur le cas `k=1`, avec start et `code(k)` initialisés à 0. La boucle `j=start+1,start+nvw(1)` devient donc une boucle `j=1,4` que nous allons détailler entièrement.

Pour `j=1`, `tablearg(1)=1` est positif, donc `sign=0`. Ensuite, on double `code(1)` et on y ajoute `sign`. On obtient donc `code(1)= 2*0 + 0 = 0`. On effectue ensuite un ishft de 5 places, mais puisque `code(1)`$= 0$, cela ne change rien. On ajoute enfin `abs(tablearg(1))=1`, ce qui donne `code(1)=1`.

Pour `j=2`, `tablearg(2)=0` est positif, donc `sign=0`. Ensuite, on double `code(1)` et on y ajoute `sign`. On obtient donc `code(1)= 2*1 + 0 = 2`. On effectue ensuite un ishft de 5 places. Puisque 2 en binaire s'écrit 10, le résultat du décalage donne 1000000 ce qui représente 64 en écriture décimale. On ajoute enfin `abs(tablearg(2))=0`, ce qui donne `code(1)=64`.

Pour `j=3`, `tablearg(3)=0` est positif, donc `sign=0`. Ensuite, on double `code(1)` et on y ajoute `sign`. On obtient donc `code(1)= 2*64 + 0 = 128`. On effectue ensuite un ishft de 5 places. Puisque 128 en binaire s'écrit 10^7 (nous utilisons la notation exponentielle par souci de facilité), le résultat du décalage donne 10^{12} ce qui représente 4096 en écriture décimale. On ajoute enfin `abs(tablearg(3))=0`, ce qui donne `code(1)=4096`.

Enfin, pour `j=4`, `tablearg(4)=1` est positif, donc `sign=0`. Ensuite, on double `code(1)` et on y ajoute `sign`. On obtient donc `code(1)= 2*4096 + 0 = 8192`. On effectue ensuite un ishft de 5 places. Puisque 8192 en binaire s'écrit 10^{13}, le résultat du décalage donne 10^{18} ce qui représente 262144 en écriture décimale. On ajoute enfin `abs(tablearg(4))=1`, ce qui donne `code(1)=262145`.

Après la sortie de la boucle sur `j`, la variable `start` est modifiée et devient `start=start+nvw(1)=4`.

Nous sortons ensuite de la boucle sur `k`, il reste alors à effectuer l'ajout de l'indicateur sinus-cosinus. Pour cela, seul `code(1)` est modifié via `code(1)=2*code(1)+sc`, nous obtenons donc finalement `code(1)= 2*262145 + 0 = 524290`.

Au final, stocker le terme $y\cos(a)$ revient donc à stocker `1.0D0` dans `tabcoef` et $(524290, 0, 0, 0, 0)$ dans `TABLE`.

Bien que cette méthode peut semble fastidieuse pour l'utilisateur, elle est en fait d'une utilité extrêmement importante dans la recherche des termes. En effet, comme nous en avons parlé à la Section 2.1, cette méthode basée sur les arbres binaires permet une recherche dichotomique des termes très rapide.

Nous allons à présent faire le descriptif complet des sous-routines présentes dans le MSNam. L'utilisateur désirant utiliser le manipulateur pourra se référer à ce descriptif pour effectuer les manipulations qu'il désire. Notons cependant que seule l'expérience permet d'exploiter pleinement les possibilités du MSNam. Le code complet du MSNam documenté plus en profondeur se trouve en Annexe B.

2.8 Quelques conventions

Dans les sous-routines du MSNam, les notations suivantes seront adoptées :

- ▷ Les variables en lettres latines majuscules correspondent aux identifiants des séries (définis comme des entiers sur 32 bits).
- ▷ Les variables en lettres grecques minuscules correspondent à des nombres réels en double précision.
- ▷ Les variables en lettres grecques majuscules correspondent à des vecteurs de nombres réels en double précision.
- ▷ Les variables en lettres latines minuscules correspondent à des nombres entiers.
- ▷ Les variables en italique correspondent à des chaînes de caractères.

De plus, rappelons également notre abus de langage lorsque nous parlons de "la série `A`", il s'agit en fait de "la série dont l'identifiant est l'entier `A`".

2.9 Sous-routines de base

Ces sous-routines ne doivent pas être appelées par l'utilisateur, et sont utilisées implicitement par le MSNam lors des manipulations des tableaux codés. Elles sont cependant indispensables au bon fonctionnement du manipulateur.
Ces sous-routines font souvent appel aux variables qui ont été définies dans le module `TABLES`.

- ▷ `INIT(A)` : Attribue un emplacement à une nouvelle série nommée `A` lorsque son premier terme est créé. Est utilisée dans les sous-routines `COPY` et `PUTTRM`.

- ▷ `MOVEBLOCK(lg,oldbeg,newbeg)` : Déplace des blocs de données dans les tables. Si nous considérons que les données sont stockées dans des cases, alors les données des cases `oldbeg` à `oldbeg+lg` seront déplacées vers les cases `newbeg` à `newbeg+lg`.
  Est utilisée dans les sous-routines `MOVE_TO_END` et `ERASE`.

- ▷ `MOVE_TO_END(A)` : déplace la série `A` à la fin de l'espace de travail.
  Est utilisée dans la sous-routine `PUTTRM`.

- ▷ `ENCODE(code,sc,arg,exp)` : Introduit dans le vecteur `code` l'indicateur "sinus-cosinus" `sc`, les arguments `arg` et exposants `exp`.
  Est utilisée dans la sous-routine `STORE`.

- ▷ `DECODE(code, sc,arg,exp)` : Opération inverse de la précédente, à savoir retirer du vecteur `code` l'indicateur "sinus-cosinus" `sc`, les arguments `arg` et exposants `exp`.
  Est utilisée dans les sous-routines `FETCH` et `SCALEP`.

- ▷ `SEARCH(A,code,left)` : Indique dans `left` l'emplacement du terme identifié par le tableau `code` dans la série `A` ou, si ce terme n'existe pas dans `A`, l'emplacement du terme qui doit le précéder immédiatement.
  Est utilisée dans la sous-routine `PUTTRM`.

- ▷ `PUTTRM(A,code,`α`)` : Ajoute à la série `A` un terme dont les arguments et exposants sont donnés dans `code` et dont le coefficient est α.
  Est utilisée dans les sous-routines `STORE` et `ACUM`.

- ▷ `ERROR` : Contient les messages d'erreurs des autres sous-routines du MSNam.
  Est utilisée par la plupart des sous-routines.

2.10 Sous-routines pour construire et afficher des séries

- ▷ `STORE(A,sc,arg,exp,`α`)` : Ajoute à la série `A` un terme dont l'indicateur "sinus-cosinus" est donné par `sc`, les arguments et exposants sont codés respectivement dans `arg` et `exp`, et dont le coefficient numérique est α.
  N.B. : la dimension de `arg` (respectivement `exp`) doit être `nvt` (respectivement `nvp`) ou plus. L'entier `sc` est 0 pour un cosinus ou 1 pour un sinus.

- ▷ `CONSTANT(A,`α`)` : Crée une série `A` d'un seul terme. L'indicateur `sc` ainsi que les arguments trigonométriques et polynomiaux sont posés à 0 et le coefficient numérique est fixé à α.

- ▷ `NBTERM(A)` : Cette fonction entière donne le nombre de termes de la série identifiée par `A`.

- ▷ `FETCH(A,j,sc,arg,exp,`α`)` : Décode le j^e terme de la série `A`, c'est-à-dire le rend lisible par l'utilisateur en passant de la représentation en code binaire à la forme décimale via l'appel à la sous-routine

DECODE. L'indicateur "sinus-cosinus" est placé dans sc, les arguments trigonométriques (respectivement polynomiaux) dans le vecteur arg (respectivement exp) et le coefficient dans α.
Cette sous-routine est habituellement appelée dans une boucle DO 1 TO NBTERM(A) pour décoder chaque terme de la série. Elle peut également être utilisée afin d'afficher les termes d'une série un par un ou dans l'ordre que l'on souhaite, contrairement à la sous-routine PRINT (voir juste après) qui écrit automatiquement tous les termes dans un fichier.
N.B. : la dimension de arg (respectivement exp) doit être nvt (respectivement nvp) ou plus. L'entier sc est 0 pour un cosinus ou 1 pour un sinus.

▷ PRINT(io,A,*label*,i) : Écrit dans le fichier FORTRAN io la série A sous une étiquette composée de la chaîne de caractère *label* et de l'entier i. L'entier io correspond au numéro du fichier ouvert par l'ulisateur. La chaîne de caractères *label*, écrite entre guillemets, renseigne l'utilisateur sur la nature de la série, par exemple "serA" ou "Hamiltonien". L'entier i n'a aucun rôle dans le cadre de PRINT, il n'a d'utilité que dans le cadre vectoriel que nous verrons plus loin, il peut donc être fixé à 1 par exemple.
La sortie consiste en une liste de termes de la forme (indicateur "sinus-cosinus", arguments, exposants, coefficient).

▷ READ(io,A) : Lit et stocke la série A à partir de l'unité désignée par io. Cette unité est supposée ouverte en lecture. Cette sous-routine suppose que le fichier a été construit par la sous-routine PRINT. Elle vérifie que les noms actuels des variables correspondent aux noms attribués lors de la création du fichier. La série A doit être une série vide en entrée, c'est-à-dire A=0.

2.11 Sous-routines pour la gestion des séries

▷ COPY(A,B) : Fait une copie de la série A et l'identifie par B. La série B doit être vie en entrée, c'est-à-dire B=0.

▷ ERASE(A) : Détruit la série A et récupère l'espace qu'elle occupait. Ses composantes sont effacées de la mémoire et l'identifiant A est 0 en sortie, indiquant que la série est vide.

▷ RENAMEE(A,B) : Échange les identifiants des séries A et B. Aucune contrainte n'est posée sur les séries A et B, et aucune des deux séries n'est effacée, seuls leurs identifiants sont échangés.

▷ CUTEPS(A,ϵ) : Efface de la série identifiée par A les termes dont le coefficient est plus petit (en valeur absolue) que ϵ.

▷ CHANGE_NAME(*oldname*,*newname*) : Substitue au nom *oldname* d'une variable le nouveau nom *newname*. Utilisé principalement avant et après l'appel à une sous-routine spécialisée (comme dev2B_fr par exemple) qui requiert des variables avec un nom bien spécifique, ou avant d'afficher quelque chose si certaines variables ont changé de signification en cours de programme.

▷ INDEXTRIG(*name*) : Fonction entière dont la valeur est le rang de la variable trigonométrique nommée *name*.

▷ INDEXPOL(*name*) : Fonction entière dont la valeur est le rang de la variable polynomiale nommée *name*.

▷ ORDER_SIZE(A) : Ordonne la série en fonction de la valeur absolue de ses coefficients (du plus grand au plus petit), c'est-à-dire des termes du vecteur tabcoef.
Cet ordre est nécessaire pour les arguments A et B de PRODCT et donc y fera appel.
Cette sous-routine peut aussi être utile avant PRINT, si l'utilisateur veut que les termes soient affichés dans cet ordre. Dans ce cas, c'est l'utilisateur qui doit prendre l'initiative de faire appel à cette sous-

routine.
N.B. : La routine change le signe de l'identifiant de la série réordonnée (il devient négatif) afin que les autres sous-routines sachent l'ordre utilisé.

▷ `ORDER_CODE(A)` : Ordonne la série en fonction de l'ordre lexicographique des "codes", c'est-à-dire des termes codés dans le tableau `TABLE`. En fait, cela correspond à classer (du plus petit au plus grand) les termes en fonction de la valeur absolue de l'argument de la première variable trigonométrique. Les ex-æquo sont classés en fonction de la valeur absolue de l'exposant de la première variable polynomiale, puis de l'argument de la deuxième variable trigonométrique, etc.
Cet ordre est nécessaire pour les séries qui peuvent potentiellement être modifiées (qui sont le résultat d'opérations telles que `STORE, ACUM, PROD, PRODCT, PDERT, PDERP`). Ces sous-routines feront automatiquement appel à celles qui permettent d'ordonner la série si cela est nécessaire.
N.B. L'identifiant de la série est positif après cette opération.

2.12 Sous-routines pour les opérations algébriques

▷ `SCALE(A,`α`)` : Multiplie les coefficients de la série `A` par α.

▷ `SCALEP(A,`*namepol*, α) : Multiplie, dans la série `A`, les coefficients de chaque terme par α à la puissance de la variable polynomiale *namepol*.

▷ `EVALP(A,`*namepol*, α) : Dans la série `A`, donne la valeur α à la variable polynomiale identifiée par *namepol*. L'exposant de la variable *namepol* est ensuite posé à 0.

▷ `EVALT(A,`*nametrig*, α) : Dans la série `A`, donne la valeur α à la variable trigonométrique identifiée par *nametrig*. L'argument de la variable trigonométrique *nametrig* est ensuite posé à 0.

▷ `XMON(A,`*namepol*, `powr`) : Multiplie la série `A` par le monôme identifié par *namepol* à la puissance `powr`.

▷ `ACUM(A,B,`α`)` : Ajoute à la série `B` le produit α`*A`. C'est-à-dire que l'on remplace `B` par son ancienne valeur à laquelle on ajoute le produit α`*A`.

▷ `PROD(A,B,C,`α`)` : Ajoute à la série `C` le produit α`*A*B`. C'est-à-dire que l'on remplace `C` par son ancienne valeur à laquelle on ajoute le produit α`*A*B`.

▷ `PRODCT(A,B,C,`α`,`ϵ`)` : Fonctionne comme la sous-routine `PROD` à l'exception près que le produit de termes individuels est annulé à chaque fois que la valeur absolue de son coefficient est inférieure à ϵ.
Le fait que `A` et `B` soient ordonnées en fonction de la valeur de leur coefficient (voir `ORDER_SIZE`) permet à la sous-routine d'éviter beaucoup de produits de termes individuels sans même en tenir compte. En considérant une possible accumulation de résultats partiels, il est conseillé de prendre un niveau de troncature ϵ bien en dessous du niveau de troncature choisi dans la sous-routine `CUTEPS` à laquelle il est d'ailleurs recommandé de faire appel après le produit.

▷ `PDERP(A,B,`*namepol*`)` : Ajoute à la série `B` la dérivée de `A` par rapport à la variable polynomiale identifiée par *namepol*.

▷ `PDERT(A,B,`*nametrig*`)` : Ajoute à la série `B` la dérivée de `A` par rapport à la variable trigonométrique identifiée par *nametrig*.

▷ `SUBSTITUTE(A,B,`*namepol*`)` : Remplace dans la série `A` la variable *namepol* par la série `B`.
La variable *namepol* ne peut pas apparaître à une puissance négative dans la série `A`.

2.13 Opérations vectorielles

Les opérations vectorielles supposent que les séries utilisées dans les appels aux sous-routines (comme `A,B,C` ci-dessous) sont en fait des vecteurs de type `A(0:order)`. Les résultats sont calculés seulement jusque `order`. La valeur de `order` doit être plus petite ou égale à `ordmax` défini dans le module `parameters`.

Une explication plus détaillée de l'intérêt et du fonctionnement des sous-routines vectorielles sera abordée ultérieurement au Chapitre 3 .

Les sous-routines suivantes ont une version vectorielle, indiqué par le "V_" devant leur nom.
Le descriptif de ces sous-routines est identique à celui présenté précédemment, avec l'argument `order` en plus.

- ▷ `V_COPY(A,B,order)`
- ▷ `V_ERASE(A,order)`
- ▷ `V_RENAMEE(A,B,order)`
- ▷ `V_CUTEPS(A,`ϵ`,order)`
- ▷ `V_PRINT(io,A,`*label*`,order)`
- ▷ `V_SCALE(A,`α`,order)`
- ▷ `V_ACUM(A,B,`α`)`
- ▷ `V_PROD(A,B,C,`α`,order)`
- ▷ `V_SUBSTITUTE(A,B,`*namepol*`,order)` : Présuppose que `A` et `B` sont des vecteurs `A(0:order)`, `B(0:order)`. La sous-routine remplace dans la série `A` la variable *namepol* par le développement jusqu'à l'ordre `order` de la série `B=`$\sum_{\texttt{k}}$`B(k)`. La variable *namepol* ne peut apparaître à une puissance négative dans la série `A`
- ▷ `V_EVALSER(`ϕ`,`Ψ`,A,order)` Voir descriptif de `EVALSER` à la section suivante.
- ▷ `V_HAMILTON(H,D,order)` Voir descriptif de `HAMILTON` à la section suivante
  Attention dans ce cas `H` est un tableau de dimension `(nvar,order)`.
- ▷ `V_POISSON(A,B,C,order)` Voir descriptif de `POISSON` à la section suivante
- ▷ `V_RK4(`$\Psi, \alpha, \beta, \eta$`,order,D)` Voir descriptif de `RK4` à la section suivante

N.B. : Une erreur fréquente pour l'utilisateur est d'oublier d'ajouter l'argument `order` lors des appels.

2.14 Opérations spécialisées

- ▷ `POWER(A,`α`,order)` : Présuppose que `A` est un vecteur `A(0:order)` où `A(k)` la composante d'ordre k est un développement en puissances d'un « petit paramètre ». En entrée, la série `A(0)` est supposée être la série unitaire $(1. * \cos(0))$.
  La sous-routine met `A` à la puissance α jusqu'à l'ordre `order` dans les puissances du petit paramètre.
- ▷ `POLAR_TO_CART(A,`*nameR,nameT,nameX,nameY*`)` : Les variables *nameR* et *nameT* sont les coordon-

nées polaires (*nameR* la distance et *nameT* l'angle) d'un point du plan, et les variables polynomiales *nameX* et *nameY* sont les coordonnées cartésiennes du même point.
La sous-routine transforme dans la série `A`, les coordonnées polaires en coordonnées cartésiennes.

▷ `CART_TO_POLAR(A,`*nameR,nameT,nameX,nameY*`)` : Les variables polynomiales *nameX* et *nameY* sont les coordonnées cartésiennes d'un point du plan, et les variables *nameR* et *nameT* sont les coordonnées polaires (*nameR* la distance et *nameT* l'angle) du même point.
La sous-routine transforme dans la série `A`, les coordonnées cartésiennes en coordonnées polaires.

▷ `dev2B_fr(A,order)` : Présuppose que deux variables polynomiales sont identifiées respectivement par ' r' et ' e' et une variable trigonométrique par ' f'. L'argument `A` en entrée est un vecteur `A(0:order)`, où `A(k)` est la composante d'ordre `k` dans le développement en puissances de ' e'. Dans le cadre du problème des deux corps, ' r' représente la distance normalisée r/a, ' e' l'excentricité et ' f' l'anomalie vraie.
La sous-routine remplace la fonction `A` par son développement en termes d'excentricité et d'anomalie moyenne. Le résultat est indépendant de la variable ' r' (c'est-à-dire que l'on fixe tous les exposants de ' r' à zéro). Dans la sortie `A`, la variable ' f' désigne l'anomalie moyenne.

▷ `BINOM(n,m)` : Fonction entière qui calcule le coefficient d'une loi binomiale, à savoir le terme

$$\left(\begin{array}{c} n \\ m \end{array} \right) = \frac{n!}{m!(n-m)!}$$

▷ `DEV_ANGLE(A,`*angle*`,pert,order)` : Substitue dans la série `A` la variable trigonométrie *angle* par l'expression `angle+pert` où `pert` est une série supposée petite. La substitution se fait par un développement en série jusqu'à l'ordre `order` de `pert`.

▷ `EVALSER(`ϕ`,`Ψ`,A)` : Sous-routine qui évalue la valeur d'une série en sommant chacun de ses terme et stocke le résultat dans un réel *phi*.
Présuppose que *Psi* est un vecteur de réels contenant les valeurs des variables polynomiales suivies par les variables trigonométriques.

▷ `HAMILTON(H,D)` : Sous-routine qui calcule les équations hamiltoniennes relatives à la série `H`, c'est-à-dire calcule les séries $\dot{p}_i =$ `namepol` et $\dot{q}_i =$ `nametrig` pour $2i =$`nvar` définies par

$$\dot{p}_i = -\frac{\partial \mathtt{H}}{\partial q_i} \quad ; \quad \dot{q}_i = \frac{\partial \mathtt{H}}{\partial p_i}. \tag{2.2}$$

Les `nvar` séries ainsi créées sont stockées dans un vecteur `D(1:nvar)` en conservant l'ordre ci-dessus, c'est-à-dire d'abord les $\dot{p}_i$ suivis des $\dot{q}_i$.

Attention, la sous-routine présuppose deux choses importantes :
– Le nombre de variables trigonométriques `nvt` doit être égal au nombre de variables polynomiales `nvp`.
– Les variables polynomiales doivent être les variables conjugués aux variables trigonométriques.

L'utilisateur qui serait dans le cas particulier où les variables ne sont pas conjuguées entre elles peut se servir de cette sous-routine comme base pour un code personnalisé.

▷ `POISSON(A,B,C)` : Sous-routine qui ajoute à la série `C` les crochets de Poisson entre entre les séries `A` et `B`.
Cette sous-routine présuppose exactement la même chose que la sous-routine `HAMILTON`.

▷ `RK4(`$\Psi, \alpha, \beta, \eta$`,D)` : Sous-routine qui effectuer l'intégration des équations hamiltoniennes contenues dans le vecteur `D` de dimension `MSnvar`. L'intégration se fait dans l'intervalle $[\alpha, \beta]$ avec un pas η. Les conditions initiales sont supposées se trouver dans le vecteur Ψ de dimension `MSnvar`.

Il est conseillé d'utiliser cette sous-routine pour des équations hamiltoniennes obtenues via l'utilisation de `HAMILTON`. En effet, ces dernières sont stockées sous la forme requise.

Pour de plus amples détails concernant les quatre dernières sous-routines, nous faisons référence au Chapitre 4.

2.15 Lecture d'une série

Il existe deux manières pour stocker les termes d'une série :

– soit en les introduisant soi-même
– soit en les lisant dans un fichier

Vu que les deux méthodes sont identiques quant à la façon de stocker la série et que la seconde est bien plus courante, nous nous concentrerons donc sur cette dernière.

En général les séries sont stockées dans des fichiers `series.dat` bruts qui contiennent sur chaque ligne les coefficients des variables.
Si nous reprenons la définition (1.24) nous aurions un fichier de données du genre :

sc	i_1	i_2	...	i_p	j_1	j_2	...	j_q	$A_{\substack{i_1,\cdots,i_p \\ j_1,\cdots,j_q}}$
0	1	2	...	-1	0	1	...	-1	1.45D-02
1	0	-1	...	2	1	-2	...	1	-4.3D-01
					⋮				

TABLE 2.1 – Exemple de fichier contenant une série

Pour stocker la série, nous allons utiliser les variables définies plus tôt pour récupérer les données, puis la sous-routine `STORE` pour les introduire dans la série, comme nous le voyons ci-dessous.

```
  open(unit=1,file='series.dat',status='old')

19 read(1,*,end=20) sc,arg,exp,coef !lecture des donnees dans le fichier

  call STORE(serA,sc,arg,exp,coef) !stockage de la serie

  goto 19

20 close(unit=1)
```

Il ne faut jamais utiliser `unit=33` pour créer un fichier, ce numéro étant utilisé par le MSNam en cas d'appel à la sous-routine `ERROR`.

NB : En général, les fichiers contenant les séries sont triés en fonction de la valeur des exposants des variables polynomiales. D'abord la somme de tous les exposants vaut 0, puis 1, puis 2, etc. Cette remarque prendra de l'importance au chapitre suivant.

2.16 Opérations et affichage des résultats

Une fois que l'utilisateur a stocké les séries dont il dispose, il peut se référer au descriptif ci-dessus pour effectuer les opérations qu'il désire. Nous ne pouvons pas donner ici un mode d'emploi exhaustif et c'est à force d'utiliser les sous-routines que l'utilisateur se rendra compte des éventuels problèmes qu'il rencontre.

La dernière chose que nous présenterons dans ce mode d'emploi concerne l'affichage des résultats, ou comment faire un fichier de sortie. En fait, il suffit de faire appel à la sous-routine `PRINT` de la manière suivante :

```
open(unit=2,file='ma_serie.dat',status='replace')
call PRINT(2,serA,'descriptif de ma serie',1)
close(unit=2)
```

NB : Puisque nous avions utilisé `unit=1` pour lire la série, nous avons arbitrairement choisi `unit=2` pour l'afficher.

A titre d'exemple, voici ce que cela donne avec une série de quelques termes.

```
      SERIES     Hamiltonien         1
       NUMBER OF TERMS :     10


      p1  p2       E1  E2             COEF
cos(   0   0 ) (   0   0)   -0.1160486010464710D-03
cos(   0   0 ) (   0   2)   -0.5672107757964912D-05
cos(   0   0 ) (   0   4)   -0.7568357084244970D-05
cos(   0   0 ) (   0   6)   -0.9879594574240365D-05
cos(   0   0 ) (   4   2)   -0.1733521571206798D-05
cos(   1  -1 ) (   1   3)    0.1379056776899787D-04
cos(   1  -1 ) (   1   5)    0.2847815345575343D-04
cos(   1  -1 ) (   3   3)    0.2166927171525686D-04
cos(   1  -1 ) (   5   1)   -0.7711335055110939D-06
cos(   2  -2 ) (   2   2)   -0.2496909744535118D-05
```

2.17 Remarques sur les erreurs

A priori, avec le descriptif et l'aide au démarrage que nous venons de fournir, tout utilisateur non expérimenté peut se lancer dans une première utilisation du MSNam.

Comme nous l'avons déjà signalé plus tôt et comme dans toute application liée à la programmation, c'est à force d'essais, d'erreurs et de modifications que l'utilisateur trouvera sa façon d'utiliser le MSNam.

Nous fournissons ici quelques clés importantes pour bien commencer et ne pas se sentir perdu avec ce programmme.

Dans le cadre d'une erreur à la compilation d'un programme, le compilateur doit permettre de résoudre les erreurs. Il arrive cependant que la compilation et l'exécution se passent normalement, mais que les résultats soient erronés. Dans ce cas, le MSNam propose une méthode de résolution interne, produite par la sous-routine `ERROR`. En effet, de nombreuses vérifications sont faites au cours de l'exécution. Si l'une d'entre elle venait

à être anormale, la sous-routine `ERROR` crée un fichier `error_log` qui explique dans quelle sous-routine est apparu le problème ainsi que les variables auxquelles il est lié. Cela permet généralement de situer et de régler ce type de problèmes.

Il existe malheureusement quelques cas insidieux qui ne sont pas détectés. Par exemple introduire un entier au lieu d'un réel en double précision dans une sous-routine peut ne provoquer aucune erreur mais produire des résultats inattendus. C'est pourquoi il est fortement conseillé de bien lire le descriptif avant d'utiliser une fonction, il peut souvent être la clé pour la résolution d'un problème malvenu.

Chapitre 3

Les sous-routines vectorielles

Comme l'utilisateur l'aura remarqué dans le chapitre précédent, la section 2.13 consacrée aux sous-routines vectorielles est très peu détaillée. En fait, les opérations vectorielles sont très utiles mais nécessitent une attention toute particulière quant à leur fonctionnement. C'est pourquoi ce chapitre sera consacré à la compréhension et à l'utilisation de ces sous-routines.

3.1 Le stockage vectoriel

Nous commencerons ce chapitre en définissant l'ordre d'un terme d'une série comme la somme des exposants de ses variables polynomiales, c'est-à-dire en référence à (2.1),

$$\text{ordre} = \sum_{i=1}^{p} i_p.$$

Cela étant, il est important d'expliquer l'intérêt de ce que nous appellerons « le stockage vectoriel ».
Il existe deux manières de stocker une série : soit en introduisant toute la série dans un seul identifiant (`A=0` dans les déclarations), soit en considérant tous les ordres possibles (`A(0:order)=0` dans les déclarations). Cette dernière méthode revient en fait à stocker `order`+1 séries au lieu d'une seule.
Du point de vue de l'affichage d'une série via la sous-routine `PRINT` (en réalité `V_PRINT`, mais nous y reviendrons), nous pouvons remarquer que cela permet de mieux se rendre compte du contenu de la série. En effet, nous avons montré à la Section 2.14 une série stockée de manière classique. Bien que celle-ci soit triée par ordre, tout est affiché d'un bloc et il peut sembler difficile de retrouver un terme particulier si la série n'était pas triée et/ou si elle contenait 10000 termes au lieu de 10.
Si nous reprenons cette même série, stockée cette fois par ordre, nous obtenons le résultat ci-dessous, beaucoup plus facile à lire.

```
      SERIES     Hamiltonien         0
       NUMBER OF TERMS :      1


      p1  p2      E1  E2             COEF
cos(   0   0 ) (   0   0)   -0.1160486010464710D-03
```

```
     SERIES     Hamiltonien         1
      NUMBER OF TERMS :      0



     SERIES     Hamiltonien         2
      NUMBER OF TERMS :      1


     p1  p2      E1  E2             COEF
cos(   0   0 ) (   0   2)   -0.5672107757964912D-05



     SERIES     Hamiltonien         3
      NUMBER OF TERMS :      0



     SERIES     Hamiltonien         4
      NUMBER OF TERMS :      3


     p1  p2      E1  E2             COEF
cos(   0   0 ) (   0   4)   -0.7568357084244970D-05
cos(   1  -1 ) (   1   3)    0.1379056776899787D-04
cos(   2  -2 ) (   2   2)   -0.2496909744535118D-05



     SERIES     Hamiltonien         5
      NUMBER OF TERMS :      0



     SERIES     Hamiltonien         6
      NUMBER OF TERMS :      5


     p1  p2      E1  E2             COEF
cos(   0   0 ) (   0   6)   -0.9879594574240365D-05
cos(   0   0 ) (   4   2)   -0.1733521571206798D-05
cos(   1  -1 ) (   1   5)    0.2847815345575343D-04
cos(   1  -1 ) (   3   3)    0.2166927171525686D-04
cos(   1  -1 ) (   5   1)   -0.7711335055110939D-06
```

Cependant, même s'il s'agit d'une commodité, l'intérêt des opérations vectorielles est tout autre. Lorsqu'on fait le développement de Taylor d'une certaine fonction, on s'arrête à un certain ordre pour une seule et bonne raison (outre le fait que nous ne pouvons pas faire de développement infini) : à partir d'un certain ordre, les termes deviennent négligeables.

Ici le principe est le même car, en mécanique céleste, beaucoup de développements se font en fonction de petits paramètres tels que l'excentricité par exemple. Dans ce cas, si les paramètres sont petits, plus nous les considérons à un ordre important, plus ils deviennent négligeables. Nous corroborerons cette remarque à l'aide d'un exemple par la suite.

Maintenant que nous avons compris l'intérêt du stockage vectoriel, il faut savoir comment l'utiliser.
Tout d'abord afin de créer des séries stockées par ordre, il faut penser à les déclarer correctement. Comme nous n'avons pas a priori de valeur de `order` bien définie, il est important de déclarer et d'initialiser les séries jusqu'à `ordmax` via une déclaration de variable du type :

```
integer:: A(0:MSordmax)=0
```

Ainsi nous allouons pour la série `A` suffisamment de place pour stocker ses termes de l'ordre 0 jusqu'à `ordmax`. Nous initialisons également chaque ordre à 0 afin de préciser que la série est vide.

En général, comme nous l'avons vu au Chapitre 2, les fichiers de données sont triés par ordre. Si tel est bien le cas, la manière classique de lire et stocker une série est la suivante :

```
program v_lecture
  use MSTABLES
  implicit none
  integer::A(0:MSordmax)=0
  integer:: sc,arg(MSnvt)=0,exp(MSnvp)=0
  integer::i,k,order
  double precision::coef
  call START_MS

  open(unit=1,file='ma_serie.dat',status='old')
  order=0
19 read(1,*,end=20) sc,arg,exp,coef !lecture des coefficients

  k=0
  do i=1,MSnvp
     k=k+exp(i) !order=somme des exposants des termes polynomiaux
  end do

  call STORE(HB(k),sc,arg,exp,coef) !stockage de la serie

  if(k.gt.order) order=k !augmentation de l'ordre si necessaire
  goto 19
20 close(unit=11)

end program v_lecture
```

On voit donc dans ce mini programme qu'à chaque lecture dans le fichier, on calcule l'ordre afin de stocker correctement la série. De plus, nous déterminons après chaque passage dans la boucle l'ordre maximal atteint par la série. Ainsi nous savons à l'avenir que travailler sur cette série peut se faire jusqu'à l'ordre `order`.

Par la suite, nous supposerons que les séries sont stockées de manière vectorielle et que les opérations sur ces séries pourront se faire jusqu'à l'ordre `order`.

3.2 Fonctionnement des sous-routines vectorielles

Nous allons maintenant voir quelle est la particularité des sous-routines vectorielles dans leur fonctionnement.

Pour la plupart de sous-routines vectorielles, il n'y en a aucune. En effet, la série n'étant plus simplement stockée d'un bloc, mais en plusieurs parties, il suffit d'effectuer l'opération « normale » sur chaque ordre. On voit donc que la plupart des sous-routines vectorielles se résument en une routine du type :

```
V_ROUTINE(...,order)
  !vérification que order < MSordmax
  do n=0,order
    call ROUTINE(...)
  end do
end V_ROUTINE
```

C'est très simple et tout à fait logique. Il suffit de penser par exemple que pour effacer une série dont on n'a plus besoin, il faut faire appel à `ERASE`. Or dans notre cas, si nous faisons `ERASE(A)`, seul le terme d'ordre 0 sera effacé. La sous-routine `V_ERASE` permet donc d'effacer chaque ordre de `A`.

Les sous-routines `V_ERASE, V_RENAMEE, V_CUTEPS, V_COPY, V_SCALE, V_ACUM, V_PRINT, V_EVALSER` fonctionnent de la sorte.

Le plus intéressant, c'est de voir ce qu'il se passe quand plusieurs ordres se mélangent. Afin de bien comprendre la chose, nous allons nous baser sur l'exemple du produit et donc de la sous-routine `V_PROD`.
Celle-ci est basée sur le schéma suivant, présenté dans le descriptif :

$$\begin{aligned}
C(0) &= C(0) + A(0) * B(0) \\
C(1) &= C(1) + A(0) * B(1) + A(1) * B(0) \\
C(2) &= C(2) + A(0) * B(2) + A(1) * B(1) + A(2) * B(0) \\
C(3) &= C(3) + A(0) * B(3) + A(1) * B(2) + A(2) * B(1) + A(3) * B(0) \\
&\vdots
\end{aligned}$$

L'idée est très simple :

- Pour calculer les termes d'ordre 0, il suffit de faire le produit des termes d'ordre 0.
- Pour les termes d'ordre 1, il faut faire le produit des termes d'ordre 0 de `A` avec ceux d'ordre 1 de `B` et inversément.
- Pour les termes d'ordre 2, il faut faire le produit des termes d'ordre 0 de `A` avec ceux d'ordre 2 de `B` et inversément, ainsi que le produit des termes d'ordre 1 de `A` et de `B`.
- etc.

Nous tenons à faire remarquer que la notation du schéma est ambigüe et que la notation `A(i)` signifie « tous les termes d'ordre i de `A` ». Le produit `A(i)*B(j)` peut donc inclure de multiples termes dont il faut faire le produit.
Cependant, dans la description et les explications ci-dessus, nous n'avons pas fait référence à l'argument `order` en lui-même. En effet, alors que la sous-routine `PROD` effectue tous les produits et ne néglige que les termes dont le coefficient est plus petit que `accuracy`, la version vectorielle `V_PROD`, en plus de négliger les termes plus petits que `accuracy`, ne fait pas les produits qui feraient apparaître des termes d'un ordre supérieur à `order`.
Par exemple, si `order` est fixé à 15, le produit d'un terme d'ordre 7 avec un terme d'ordre 9 (qui produirait un terme d'ordre 16) n'est pas calculé. Il faut cependant être sûr que les paramètres de la série sont suffisamment petits pour être négligés à un certain ordre.

Afin d'illustrer les résultats de cette section, nous allons nous baser sur un exemple concret. Les deux séries que nous allons multiplier sont les séries A et B de l'Annexe C, tronquée à l'ordre 4.

Le produit de ces deux séries de manière classique donne le résultat suivant :

```
      SERIES     Product          1
       NUMBER OF TERMS :     20


      p1  p2       E1  E2             COEF
cos(   0   0 ) (   2   0)    0.6582401702966634D-09
cos(   0   0 ) (   2   2)    0.1599058827449046D-08
cos(   0   0 ) (   2   4)    0.1512171802203878D-09
cos(   0   0 ) (   2   6)    0.1021878145064947D-09
cos(   0   0 ) (   4   0)   -0.9485424133123699D-10
cos(   0   0 ) (   4   2)   -0.4636190986182907D-11
cos(   0   0 ) (   4   4)    0.2664629185206423D-10
cos(   1  -1 ) (   1   1)   -0.5335819769948339D-09
cos(   1  -1 ) (   1   3)   -0.2607988759822010D-10
cos(   1  -1 ) (   1   5)   -0.3479868692253478D-10
cos(   1  -1 ) (   3   1)   -0.5525741544047912D-09
cos(   1  -1 ) (   3   3)   -0.1109700481316411D-09
cos(   1  -1 ) (   3   5)   -0.2222372880226029D-09
cos(   1  -1 ) (   5   3)    0.5327336498024201D-11
cos(   2  -2 ) (   2   4)    0.3170395139496981D-10
cos(   2  -2 ) (   4   2)    0.1416274113291583D-10
cos(   2  -2 ) (   4   4)    0.6654564179755946D-10
cos(   2  -2 ) (   6   2)   -0.2040890431721871D-11
cos(   3  -3 ) (   3   3)   -0.5740293402301330D-11
cos(   3  -3 ) (   5   3)   -0.5944611905140810D-11
```

Si nous effectuons le produit avec `V_PROD` en ne tronquant pas la série produit (dans ce cas en choisissant `order` > 8), nous obtenons alors ce résultat :

```
      SERIES     Product order         0
       NUMBER OF TERMS :      0
```

```
     SERIES     Product order         1
      NUMBER OF TERMS :      0


     SERIES     Product order         2
      NUMBER OF TERMS :      2

     p1  p2      E1  E2             COEF
cos(   0   0 ) (   2   0)    0.6582401702966634D-09
cos(   1  -1 ) (   1   1)   -0.5335819769948339D-09


     SERIES     Product order         3
      NUMBER OF TERMS :      0


     SERIES     Product order         4
      NUMBER OF TERMS :      4

     p1  p2      E1  E2             COEF
cos(   0   0 ) (   2   2)    0.1599058827449046D-08
cos(   0   0 ) (   4   0)   -0.9485424133123699D-10
cos(   1  -1 ) (   1   3)   -0.2607988759822010D-10
cos(   1  -1 ) (   3   1)   -0.5525741544047912D-09


     SERIES     Product order         5
      NUMBER OF TERMS :      0


     SERIES     Product order         6
      NUMBER OF TERMS :      7

     p1  p2      E1  E2             COEF
cos(   0   0 ) (   2   4)    0.1512171802203877D-09
cos(   0   0 ) (   4   2)   -0.4636190986182907D-11
cos(   1  -1 ) (   1   5)   -0.3479868692253478D-10
cos(   1  -1 ) (   3   3)   -0.1109700481316411D-09
cos(   2  -2 ) (   2   4)    0.3170395139496981D-10
cos(   2  -2 ) (   4   2)    0.1416274113291583D-10
cos(   3  -3 ) (   3   3)   -0.5740293402301330D-11
```

```
      SERIES     Product order         7
       NUMBER OF TERMS :      0



      SERIES     Product order         8
       NUMBER OF TERMS :      7


      p1  p2       E1  E2             COEF
cos(   0   0 ) (   2   6)    0.1021878145064947D-09
cos(   0   0 ) (   4   4)    0.2664629185206423D-10
cos(   1  -1 ) (   3   5)   -0.2222372880226029D-09
cos(   1  -1 ) (   5   3)    0.5327336498024201D-11
cos(   2  -2 ) (   4   4)    0.6654564179755946D-10
cos(   2  -2 ) (   6   2)   -0.2040890431721871D-11
cos(   3  -3 ) (   5   3)   -0.5944611905140810D-11
```

Le lecteur pourra vérifier que les deux résultats concordent, et pourra constater que la lecture est bien plus aisée avec la version vectorielle dans ce cas.

Si maintenant nous décidons de tronquer le résultat du produit, par exemple à l'ordre 6, nous obtenons les résultats suivants :

```
      SERIES     Product order         0
       NUMBER OF TERMS :      0



      SERIES     Product order         1
       NUMBER OF TERMS :      0



      SERIES     Product order         2
       NUMBER OF TERMS :      2


      p1  p2       E1  E2             COEF
cos(   0   0 ) (   2   0)    0.6582401702966634D-09
cos(   1  -1 ) (   1   1)   -0.5335819769948339D-09
```

```
      SERIES     Product order         3
       NUMBER OF TERMS :      0



      SERIES     Product order         4
       NUMBER OF TERMS :      4


      p1  p2       E1  E2             COEF
cos(   0   0 ) (   2   2)    0.1599058827449046D-08
cos(   0   0 ) (   4   0)   -0.9485424133123699D-10
cos(   1  -1 ) (   1   3)   -0.2607988759822010D-10
cos(   1  -1 ) (   3   1)   -0.5525741544047912D-09



      SERIES     Product order         5
       NUMBER OF TERMS :      0



      SERIES     Product order         6
       NUMBER OF TERMS :      7


      p1  p2       E1  E2             COEF
cos(   0   0 ) (   2   4)    0.1512171802203877D-09
cos(   0   0 ) (   4   2)   -0.4636190986182907D-11
cos(   1  -1 ) (   1   5)   -0.3479868692253478D-10
cos(   1  -1 ) (   3   3)   -0.1109700481316411D-09
cos(   2  -2 ) (   2   4)    0.3170395139496981D-10
cos(   2  -2 ) (   4   2)    0.1416274113291583D-10
cos(   3  -3 ) (   3   3)   -0.5740293402301330D-11
```

Dans ce cas, nous pouvons voir que les premiers termes sont égaux à ceux obtenus précédemment, mais que tous les termes d'un ordre supérieur à 6 ont été négligés.

Remarque : Cette série ayant été créée de manière quelque peu artificielle, les résultats obtenus n'ont aucune signification scientifique. C'est pourquoi si nous voulions évaluer la série avec certaines valeurs des paramètres, mêmes petites, la série tronquée donnerait des résultats sensiblement différents de ceux de la série entière.
C'est pourquoi nous conseillerons toujours à l'utilisateur qui voudrait tronquer une série, d'essayer (si possible) d'abord le produit jusqu'à `ordmax` pour ensuite vérifier la sensibilité du problème considéré à la troncature.

3.3 Autres subtilités des opérations vectorielles

Dans cette section nous commenterons brièvement certaines subtilités des opérations vectorielles.

Nous insistons d'abord sur l'importance de déclaration les séries sous la forme `A(0:order)=0` et pas

`A(1:order)=0` ou via l'utilisation de `dimension(order)`.

En effet, les sous-routines vectorielles ont toutes été codées en faisant des boucles sur l'ordre allant de 0 à `order`. Leur utilisation sur un vecteur ne commençant pas à l'ordre 0 pourrait donc produire des résultats inopinés allant d'un simple décalage dans l'ordre jusqu'à des calculs totalement faux ou un interruption anormale du programme dans le pire des cas.

Par ailleurs, tant que nous parlons de décalage dans l'ordre, nous pouvons mentionner le problème de la dérivation d'une série. En effet, lorsque nous dérivons une série par rapport à une variable polynomiale (via `PDERP`), le résultat décroît d'un ordre par la même occasion. Il est donc important de penser à deux choses lors de l'utilisation de `PDERP` sur un vecteur. D'une part il faut stocker la dérivée à l'ordre inférieur via un appel du type :

`call PDERP(A(k),Ader(k-1),`*namepol*`)`.

D'autre part il ne faut commencer les dérivées qu'à l'ordre 1. En effet, bien que le MSNam peut dériver les termes d'ordre 0 (cela vaut 0 bien sûr), il tenterait de les stocker à l'ordre -1 à cause de la remarque ci-dessus.

Lors de ce travail, nous avons pensé automatiser cela via la création d'une sous-routine `V_PDERP`. Cependant, vu que le problème ne se pose pas avec `PDERT`, cela rendrait déroutant l'existence d'une seule des deux opérations de dérivation en version vectorielle. C'est pourquoi nous avons décidé de ne pas l'implémenter mais d'en informer l'utilisateur.

Une autre remarque concernant les opérations vectorielles est qu'elles sont plus gourmandes en mémoire, plus précisément concernant le nombre total de séries utilisées. En effet, puisque chaque série n'est plus stockée en une fois, mais en maximum `ordmax`$+1$ fois, le nombre de séries utilisables en même temps a drastiquement diminué. Nous en profitons donc pour rappeller à l'utilisateur l'importance d'effacer les séries obsolètes avec `ERASE` (enfin, `V_ERASE` dans ce cas).

La dernière remarque concerne la sous-routine `V_PRINT` qui est la seule à ne pas avoir d'argument d'entrée en plus que sa version classique. Nous rappelons que la sous routine `PRINT(io,A,`*label*`,i)` écrit dans le fichier `io` la série `A` sous une étiquette composée de la chaîne de caractères *label* et de l'entier `i`. En fait, l'entier `i` n'a aucune utilité dans le cas non vectoriel, nous le fixons d'ailleurs généralement à 1.
La sous-routine `V_PRINT(io,A,`*label*`,order)` va, comme nous l'avons dit précédemment, faire appel `order+1` fois à `PRINT` et cette fois l'entier `i` correspond à l'ordre, comme nous avons pu le voir à la section 3.1.

Chapitre 4

Améliorations apportées au code

Dans ce chapitre, nous décrirons d'abord les améliorations apportées au code de J. Henrard (2004). L'objectif est d'une part de rendre le programme plus facile à lire et à comprendre, sans devoir toujours revenir au descriptif des chapitres précédents et, d'autre part, de corriger certaines imperfections. Ensuite, afin d'adapter plus encore le MSNam à son usage en mécanique céleste, nous avons créé de nouvelles sous-routines qui seront décrites ici.

4.1 Modifications de l'ancien code

4.1.1 En-têtes et commentaires

Le MSNam est pauvre en commentaires, et les sous-routines n'ont pas d'en-tête au sein même du code. Il semble essentiel de remédier à ce problème en introduisant dans chaque sous-routine une en-tête décrivant ce que fait cette partie du programme et quels sont les arguments utilisés. De plus, seuls les commentaires absolument nécessaires à la compréhension d'une opération bien particulière étaient présents dans le MSNam. C'est pourquoi des commentaires supplémentaires ont été ajoutés au sein des lignes de code pour expliquer certaines parties plus en profondeur.

Outre l'ajout des en-têtes, les principaux commentaires qui ont été ajoutés concernent les parties faisant appel à une vérification et à la sous-routine `ERROR`. Nous avons introduit un commentaire expliquant quelle vérification le MSNam fait à ce moment précis. Nous pouvons prendre par exemple la sous-routine `NBTERM` telle qu'elle était à l'origine :

```
! ******************************************************************
function NBTERM(ser)
! ******************************************************************
use MSTABLES
implicit none
integer,intent(in):: ser
integer::NBTERM,absser

! Validation
absser=abs(ser)
if(absser.gt.MSnmaxser) then
MSmessage='error in NBTERM: the identifier (1) is&
& not valid '
MSintmess=0
```

```
MSintmess(1)=ser
call ERROR
endif

! number of terms

if(ser.eq.0) then
NBTERM=0
else
NBTERM=MSlength(absser)
endif

end function NBTERM
```

Avec l'ajout des commentaires et de l'en-tête, la sous-routine est devenue :

```
!*********************************************************************!
! function NBTERM(ser)                                                !
! The integer function NBTERM gives the number of terms of the series  !
! identified by ser                                                   !
!                                                                     !
!*********************************************************************!
function NBTERM(ser)
  use MSTABLES
  implicit none
  integer,intent(in):: ser
  integer::NBTERM,absser

  ! Validation
  absser=abs(ser)
  if(absser.gt.MSnmaxser) then ! if there are more series than permitted
     MSmessage='error in NBTERM: the identifier (1) is&
          & not valid '
     MSintmess=0
     MSintmess(1)=ser
     call ERROR
  endif

  ! number of terms

  if(ser.eq.0) then ! if the series is empty
     NBTERM=0
  else
     NBTERM=MSlength(absser)
  endif

end function NBTERM
```

Cela peut sembler assez futile dans le cas d'une fonction aussi simple, mais dès que l'on passe à un niveau de difficulté plus élevé, l'ajout de commentaires peut s'avérer très utile.

4.1.2 Uniformisation du code

Certaines fonctions similaires telles que `EVALP` et `EVALT` ont été codées avec des arguments différents, sans aucune raison apparente. En effet, l'aide du MSNam décrit ces deux sous-routines de manière quasi identique, alors que la définition de la sous-routine au sein du code en FORTRAN est différente.
Nous pouvons constater ci-dessous qu'avant même d'effectuer une quelconque modification, la sous-routine `EVALP` correspond au descriptif du chapitre précédent. Les arguments en entrée sont les mêmes que ceux de la Section 2.12 du descriptif (sauf `A` qui a été modifié en `ser` presque partout).

```
!*****************************************
subroutine EVALP(ser,namepol,alpha)
!*****************************************
  use MSTABLES
  implicit none
  double precision,intent(in)::alpha
  integer,intent(in):: ser
  integer:: sc,arg(MSnvt),exp(MSnvp),temp=0,NBTERM
  integer::j,k,absser,indexpol
  double precision:: coef
  character(len=4),intent(in):: namepol


  !code de la sous-routine

end subroutine EVALP
```

Cependant, les arguments en entrée de la sous-routine `EVALT` ne correspondent pas exactement à ceux du descriptif comme nous pouvons le constater ci-dessous (nous n'affichons ici que le début et la fin du code afin d'alléger l'écriture)

```
! ********************************************************************
SUBROUTINE EVALT(ser,name,value)
! ********************************************************************
  use MSTABLES
  implicit none
  integer,intent(inout):: ser
  character(len=4),intent(in)::name
  double precision,intent(in):: value
  integer:: CoCo=0,CoSi=0,SiCo=0,SiSi=0,Res=0
  integer:: sc,arg(MSnvt),exp(MSnvp),NBTERM
  integer:: i,imax,j,INDEXTRIG,indT
  double precision:: coef

  !code de la sous-routine

end subroutine EVALT
```

Techniquement, les arguments d'entrée d'une sous-routine sont des arguments muets, et leur donner tel ou tel nom ne pose donc aucun problème. Nous estimons cependant qu'étant donné qu'un descriptif existe, il est

préférable de suivre les notations qui y apparaissent afin de ne pas compliquer l'utilisation du programme.
De plus, nous avons constaté que l'argument d'entrée nommé `value` est en fait reconnu par le compilateur IFORT comme un paramètre intrinsèque à FORTRAN (c'est-à-dire que `value` est en fait une fonction reconnue par FORTRAN et qui permet de dire qu'un argument est passé par valeur). Bien que cela n'empêche en rien le MSNam de fonctionner, il est préférable de ne jamais utiliser comme variable un nom utilisé par un langage de programmation. C'est pourquoi nous avons décidé de renommer la variable `value` par `alpha` (ce qui uniformise l'écriture vis-à-vis du descriptif).
Nous pouvons également constater que dans les commentaires existants, il est question de l'argument `nametrig`. Or, `nametrig` est le nom qui a été donné à cette variable dans le descriptif, mais au sein du code, elle été déclarée comme étant `name`. Nous remarquons donc un problème évident d'uniformisation du code en comparant cette sous-routine avec son descriptif, et en faisant le parallèle avec la sous-routine `EVALP`. Nous avons donc remplacé le nom de cette variable.

Nous avons constaté que la déclaration d'une variable nommée `result` dans les sous-routines `PROD, PRODCT, PDERT` et `PDERP` pose le même souci que la variable `value` dans ces sous-routines, car elle correspond à un type utilisé par FORTRAN (`result` est un argument utilisé en FORTRAN qui permet de déclarer de manière différente de l'habitude une fonction à valeur numérique).

4.1.3 Correction du descriptif

Nous avons constaté que l'ancien descriptif du MSNam [Aide MSNam] décrit une sous-routine nommée `INDEXARG`. Cependant, cette sous-routine n'existe pas, il s'agit en fait de `INDEXTRIG`. De plus, certaines sous-routines ont été ajoutées au MSNam depuis la parution de [Aide MSNam] et n'en font donc pas partie. Elles ont donc été ajoutées à la Section 2.14 : "Opérations spécialisées".

4.1.4 Correction du code

Lors de la lecture du code, nous avons constaté à plusieurs endroits que dans l'ancienne version du MSNam, la manière dont les arguemnts d'entrée passaient n'était pas spécifiée. Heureusement sans conséquence, nous avons en effet constaté qu'il manquait parfois un `INTENT(IN)` pour certains arguments de sous-routines. Sans conséquence car, par défaut en FORTRAN, un argument dont l'`INTENT` n'a pas été précisé passe automatique en `INOUT`. Il y aurait cependant pu y avoir un souci si cet argument avait été modifié au cours de la sous-routine, mais ce n'était pas le cas.

4.2 Ajout de nouvelles sous-routines

Dans cette section, nous détaillerons les sous-routines qui ont été rajoutées au MSNam ainsi que les subtilités qui y sont liées.

4.2.1 L'évaluation d'une série

En premier lieu, nous avons remarqué qu'il n'existait aucun moyen automatisé d'évaluer la valeur d'une série. Comme nous le verrons au Chapitre 6 dans un exemple concret, il est souvent nécessaire de pouvoir évaluer une série dont on connaît la valeur des paramètres. C'est pourquoi nous avons mis en place une sous-routine très simple basée sur le principe suivant :

Arguments d'entrée :	ser	la série que l'on veut évaluer
	values	un vecteur de dimension `nvar` contenant la valeur des paramètres
Arguments de sortie :	f	la valeur de la série évaluée

L'évaluation d'une série suit un processus très simple. Nous effectuons une boucle sur le nombre total de termes de la série, et décodons d'abord chaque terme. Ensuite, nous commençons par calculer la valeur des variables polynomiales à la puissance voulue et les multiplions entre elles. Puis nous calculons la somme des valeurs des variables trigonométriques multipliées par leur coefficient respectif. Il suffit ensuite de vérifier s'il s'agit d'un sinus ou d'un cosinus, et d'effectuer le produit du tout. Nous obtenons ainsi le résultat voulu.

En FORTRAN, cela donne le résultat suivant :

```
!*************************************************************************!
!subroutine EVALSER(f,values,ser)                                         !
!The subroutine EVALSER calculates the value of the series ser. The values   !
!of the parameters are supposed to be stored in values (polynomial variables !
!and then trigonometric ones) and the solution is stored in f.               !
!                                                                         !
!*************************************************************************!
SUBROUTINE EVALSER (f,values,ser)
  use MSTABLES
  implicit none
  double precision, INTENT(INOUT)::f
  double precision, INTENT(IN)::values(MSnvar)
  integer, INTENT(IN)::ser
  integer::i,j,NBTERM,absser
  integer::sc,exp(MSnvp),arg(MSnvt)
  double precision::coef,term,trig

  !Nothing to do if ser is empty
  if(ser.eq.0) then
     f=0.0d0
     return
  end if

  ! Validation
  absser=abs(ser)
  if(absser.gt.MSnmaxser) then ! if there are more series than permitted
     MSmessage='error in EVALSER : the identifier (1) is&
          & not valid '
     MSintmess=0
     MSintmess(1)=ser
     call ERROR
  endif

  f=0.0D0 !Initialisation
  do j=1,NBTERM(ser) ! sum over all terms
     call FETCH(ser,j,sc,arg,exp,coef) ! decode term number j
     term=1.D0
     trig=0.D0
     do i=1,MSnvp ! make product of polynomial variables
        term=term*values(i)**exp(i)
     end do
     do i=1,MSnvt ! make sum of trigonometric arguments
        trig=trig+arg(i)*values(i+MSnvp)
```

```
     end do
     if (sc==0) then ! if cosine
        f=f+coef*term*cos(trig)
     else  ! if sine
        f=f+coef*term*sin(trig)
     end if
  end do

END SUBROUTINE EVALSER
```

Nous avons également implémenté la version vectorielle de cette sous-routine, et faisons référence au Chapitre 3 pour de plus amples informations.

```
!*****************************************************************************!
!subroutine V_EVALSER(f,values,ser,order)                                     !
!The subroutine V_EVALSER is the same as EVALSER for vectorial operations     !
!                                                                             !
!*****************************************************************************!
SUBROUTINE V_EVALSER (f,values,ser,order)
  use MSTABLES
  implicit none
  double precision, INTENT(INOUT)::f
  double precision, INTENT(IN)::values(MSnvar)
  integer, INTENT(IN)::order
  integer,INTENT(IN)::ser(0:order)
  integer::k
  double precision::ftmp

  ! validation of "order"
  if(order.lt.0.or.order.gt.MSordmax) then
     MSmessage='error in V_EVALSER: the order (1) is out of bound'
     MSintmess(1)=order
     call ERROR
  endif

  f=0.0D0 !Initialisation
  do k=0,order ! calcul par ordre
     ftmp=0.d0
     call EVALSER(ftmp,values,ser(k))
     f=f+ftmp
  end do

END SUBROUTINE V_EVALSER
```

4.2.2 Les équations d'Hamilton

Ensuite, nous avons également implémenté une sous-routine permettant de calculer les équations d'Hamilton. Supposons que la série de Poisson considérée corresponde à l'hamiltonien du système que nous désirons étudier.

L'hamiltonien $\mathcal{H}(q_i, p_i)$ dépend des coordonnées généralisées $q_i (i = 1, \ldots, n)$ et des moments conjugués $p_i (i = 1, \ldots, n)$, les équations hamiltoniennes sont données par

$$\dot{p_i} = -\frac{\partial \mathcal{H}}{\partial q_i} \quad ; \quad \dot{q_i} = \frac{\partial \mathcal{H}}{\partial p_i} \quad \text{pour } i = 1, \ldots, n. \tag{4.1}$$

Alors `nvar` = `nvt` et nous pouvons identifier les variables trigonométriques `namevt` aux q_i et les variables polynomiales aux p_i.

En fait, après toutes les vérifications d'usage, la sous-routine se résume à quatre appels aux sous-routines de dérivation. Nous allons créer un vecteur `deriv` de dimension `nvar` qui contiendra les séries correspondants aux équations hamiltoniennes. En respectant nos conventions, nous calculons donc d'abord les dérivées relatives aux variables trigonométries avec `PDERT`, c'est-à-dire la partie $\frac{\partial \mathcal{H}}{\partial q_i}$. Il faut ensuite penser à multiplier le coefficient par -1, ce qui se fait via l'appel à `SCALE`.
Ensuite, on calcule les dérivées relatives aux variables polynomiales avec `PDERP`, c'est-à-dire la partie $\frac{\partial \mathcal{H}}{\partial p_i}$.
Les `nvar` séries résultantes stockées dans `deriv` sont donc ordonnées comme suit : $(\dot{p_1}, \dot{p_2}, \ldots, \dot{p_n}, \dot{q_1}, \dot{q_2}, \ldots, \dot{q_n})$.
Nous obtenons ainsi le code suivant :

```
!*****************************************************************************!
!subroutine HAMILTON(ser,deriv)                                               !
!The subroutine HAMILTON calculates the hamiltonian equations associated to   !
!the Hamiltonian ser, with angular variables namevt and polynomial variables  !
!namevp.                                                                      !
!The output is the vector deriv where the Hamiltonian equations are ordered   !
!as (p1',p2',...,q1',q2',...)                                                 !
!                                                                             !
!*****************************************************************************!
SUBROUTINE HAMILTON(ser,deriv)
  use MSTABLES
  implicit none
  integer, INTENT(IN)::ser
  integer, dimension(MSnvar), INTENT(INOUT)::deriv
  integer::i,absser

  !Nothing to do if ser is empty
  if(ser.eq.0) return

  ! Validation
  absser=abs(ser)
  if(absser.gt.MSnmaxser) then ! if there are more series than permitted
     MSmessage='error in HAMILTON : the identifier (1) is&
          & not valid '
     MSintmess=0
     MSintmess(1)=ser
     call ERROR
  endif

  !Check if nvt=nvp
```

```
  if(MSnvt.ne.MSnvp) then
     MSmessage='error in HAMILTON : argument (1) is not equal &
          & to (2)'
     MSintmess=0
     MSintmess(1)=MSnvt
     MSintmess(2)=MSnvp
     call ERROR
  endif

  !Check if deriv is empty
  do i=1,MSnvar
     if(abs(deriv(i)).ne.0) then
        MSmessage='error in HAMILTON : the series (1) is not empty'
        MSintmess=0
        MSintmess(1)=deriv(i)
     endif
  end do

  !Calculation of the Hamiltonian equations
  do i=1,MSnvt
     call PDERT(ser,deriv(i),MSnamevt(i)) !dH/dq
     call SCALE(deriv(i),-1.d0) !p'=-dH/dq
     call PDERP(ser,deriv(i+MSnvt),MSnamevp(i)) !q'=dH/dp
  end do

END SUBROUTINE HAMILTON
```

Nous avons également implémenté la version vectorielle de cette sous-routine, et faisons référence au Chapitre 3 pour de plus amples informations. Nous faisons seulement remarquer que la sortie `deriv` n'est plus un vecteur de dimension `nvar` mais un tableau de dimension `nvar,order`.

```
SUBROUTINE V_HAMILTON(ser,deriv,order)
!*************************************************************************!
!subroutine V_HAMILTON(ser,deriv,order)                                   !
!The subroutine V_HAMILTON is the same as HAMILTON for vectorial operations  !
!                                                                         !
!*************************************************************************!
  use MSTABLES
  implicit none
  integer, INTENT(IN)::order
  integer, dimension(order), INTENT(IN)::ser
  integer, dimension(MSnvar,order), INTENT(INOUT)::deriv
  integer::i,k


  ! validation of "order"
  if(order.lt.0.or.order.gt.MSordmax) then
     MSmessage='error in V_HAMILTON : the order (1) is out of bound'
     MSintmess(1)=order
     call ERROR
```

```
  endif

  do k=0,order
     if(abs(ser(k)).gt.MSnmaxser) then ! if there are more series than permitted
        MSmessage='error in V_HAMILTON : the identifier of ser(index (1)) &
             & is (2) and is not valid '
        MSintmess=0
        MSintmess(1)=k
        MSintmess(2)=ser(k)
        call ERROR
     endif
  enddo


  !Check if nvt=nvp
  if(MSnvt.ne.MSnvp) then
     MSmessage='error in V_HAMILTON : argument (1) is not equal &
          & to (2)'
     MSintmess=0
     MSintmess(1)=MSnvt
     MSintmess(2)=MSnvp
     call ERROR
  endif

  !Check if deriv is empty
  do i=1,MSnvar
     do k=0,order
        if(abs(deriv(i,k)).gt.MSnmaxser) then ! if there are more series than permitted
           MSmessage='error in V_HAMILTON : the identifier of deriv(inder(1),index (2)) &
                & is (3) and is not valid '
           MSintmess=0
           MSintmess(1)=i
           MSintmess(2)=k
           MSintmess(3)=ser(k)
           call ERROR
        endif
     end do
  end do


  do k=0,order
     do i=1,MSnvt
        call PDERT(ser(k),deriv(i,k),MSnamevt(i)) !dH/dq
        call SCALE(deriv(i,k),-1.d0) !-dH/dq=p'
        !For polynomial derivative, remark than we store in order k-1
        if(k>0) then
           call PDERP(ser(k),deriv(i+MSnvt,k-1),MSnamevp(i)) !q'=dH/dp
        end if
     end do
  end do

END SUBROUTINE V_HAMILTON
```

4.2.3 Les crochets de Poisson

En mécanique hamiltonienne et notamment dans la théorie des perturbations via les séries de Lie, apparaissent souvent ce qu'on appelle les crochets ou parenthèses de Poisson.

Nous rappelons que dans le cadre de deux fonctions f et g dépendant des $2n$ variables (q_i, p_i) pour $i = 1, \ldots, n$, les crochets de Poisson sont définis par

$$\{f, g\} = \sum_{i=1}^{n} \frac{\partial f}{\partial q_i}\frac{\partial g}{\partial p_i} - \frac{\partial f}{\partial p_i}\frac{\partial g}{\partial q_i} \tag{4.2}$$

Dans notre cas, f et g consistent en deux séries `A` et `B`

Bien que l'utilisation des crochets de Poisson se fasse régulièrement dans les domaines que nous traitons, il n'existait pas de sous-routine permettant de les calculer directement dans l'ancienne version du MSNam. C'est pourquoi nous avons décidé de mettre en place cette nouvelle sous-routine automatisée.

Comme vous pouvez le constater dans le code ci-dessous, l'unique restriction de la sous-routine est que `nvt` soit égal à `nvp` afin de correspondre aux définitions ci-dessus. En fait, nous avons fait les mêmes hypothèses que pour les équations hamiltoniennes de la Section 5.2 concernant le nombre de variables et la correspondance de celles-ci avec p_i et q_i.

Concrètement, elle est assez simple à créer puisqu'elle ne fait appel qu'à des dérivées et des produits de séries. Il suffit d'abord de calculer les deux premières dérivées et d'en faire le produit. Pour rappel, la sous-routine `PROD` fait non seulement le produit de `A` et `B`, mais somme également ce produit dans `C` avec un éventuel coefficient. Ici, nous initialisons d'abord le résultat à une série vide, et nous ajoutons le premier produit (coefficient = 1). Ensuite, nous calculons les deux dérivées suivantes et en faisons le produit que nous retirons du résultat (coefficient = -1). Il suffit de répéter l'opération `nvp` (ou `nvt`) fois en n'oubliant pas de réinitialiser les séries intermédiaires.

```
!*****************************************************************************!
!subroutine POISSON(serA,serB,res)                                            !
!The subroutine POISSON calculates the Poisson brackets between serA and serB !
!and add the result in res.                                                   !
!It is supposed that nvt and nvp are equal.                                   !
!In case of an Hamiltonian in variables q and momentum p, we assume           !
!that the polynomial (resp. trigonometric) variables namevp (resp. namevt)    !
!are the momentum p (resp. the variables q).                                  !
!                                                                             !
!*****************************************************************************!
SUBROUTINE POISSON(serA,serB,res)
  use MSTABLES
  implicit none
  integer, INTENT(IN)::serA,serB
  integer, INTENT(OUT)::res
  integer::Aq=0,Ap=0,Bq=0,Bp=0
  integer::i
  integer::absserA,absserB

  ! Nothing to do if A or B are null
  if(serA.eq.0.or.serB.eq.0) then
     return
  end if
```

```
! Validation
absserA=abs(serA)
if(absserA.gt.MSnmaxser) then ! if there are more series than permitted
   MSmessage='error in POISSON: the identifier (1) of the &
        &first series is not valid '
   MSintmess=0
   MSintmess(1)=serA
   call ERROR
endif

absserB=abs(serB)
if(absserB.gt.MSnmaxser) then ! if there are more series than permitted
   MSmessage='error in POISSON: the identifier (1) of the &
        &second series is not valid '
   MSintmess=0
   MSintmess(1)=serB
   call ERROR
endif

if(abs(res).gt.MSnmaxser) then ! if there are more series than permitted
   MSmessage='error in POISSON: the identifier (1) of the &
        & series res is not valid '
   MSintmess=0
   MSintmess(1)=res
   call ERROR
endif


!Check if nvt=nvp
if(MSnvt.ne.MSnvp) then
   MSmessage='error in POISSON : argument (1) is not equal &
        & to (2)'
   MSintmess=0
   MSintmess(1)=MSnvt
   MSintmess(2)=MSnvp
   call ERROR
endif


do i=1,MSnvt !Sum over the number of variables
   call PDERT(serA,Aq,MSnamevt(i)) !Calculate dA/dq
   call PDERP(serB,Bp,MSnamevp(i)) !Calculate dB/dp
   call PROD(Aq,Bp,res,1.D0) !Make product and add to res
   call ERASE(Aq)
   Call ERASE(Bp)
   call PDERP(serA,Ap,MSnamevp(i)) !Calculate dA/dp
   call PDERT(serB,Bq,MSnamevt(i)) !Calculate dB/dq
   call PROD(Ap,Bq,res,-1.D0) !Make product and substract from res
   call ERASE(Ap)
   call ERASE(Bq)
end do
```

```
END SUBROUTINE POISSON
```

Afin de vérifier que la sous-routine fonctionne, nous avons décidé de l'appliquer sur un exemple trivial, que nous allons d'abord résoudre à la main.

Soient les deux séries A et B définies ci-dessous dans les variables polynomiales x, y et les variables trigonométriques a, b.

$$\begin{aligned} A &= 1.2 + 2x^3 + 3xy\cos a - xy^2\sin(b-a) \\ B &= x^2\cos(2a+3b) + 2xy\sin(3a-2b) \end{aligned}$$

Dans ce cas précis, les parenthèses de Poisson définies en (4.2) s'écrivent sous la forme

$$\begin{aligned} \{A,B\} &= \sum_{i=1}^{2} \frac{\partial A}{\partial q_i}\frac{\partial B}{\partial p_i} - \frac{\partial A}{\partial p_i}\frac{\partial B}{\partial q_i} \\ &= \frac{\partial A}{\partial a}\frac{\partial B}{\partial x} - \frac{\partial A}{\partial x}\frac{\partial B}{\partial a} + \frac{\partial A}{\partial b}\frac{\partial B}{\partial y} - \frac{\partial A}{\partial y}\frac{\partial B}{\partial b} \end{aligned} \tag{4.3}$$

Nous calculons ensuite chacune des dérivées intervenant dans la définition (4.2), ce qui donne les résultats suivants :

$$\begin{aligned} \frac{\partial A}{\partial x} &= 6x^2 + 3y\cos a - y^2\sin(b-a) \\ \frac{\partial A}{\partial y} &= 3x\cos a - 2xy\sin(b-a) \\ \frac{\partial A}{\partial a} &= -3xy\sin a + xy^2\sin(b-a) \\ \frac{\partial A}{\partial b} &= -xy^2\cos(b-a) \\ \frac{\partial B}{\partial x} &= 2x\cos(2a+3b) + 2y\sin(3a-2b) \\ \frac{\partial B}{\partial y} &= 2x\sin(3a-2b) \\ \frac{\partial B}{\partial a} &= -2x^2\sin(2a+3b) + 6xy\cos(3a-2b) \\ \frac{\partial B}{\partial b} &= -3x^2\sin(2a+3b) - 4xy\cos(3a-2b) \end{aligned} \tag{4.4}$$

Grâce aux formules de Simpson, les produits suivants sont immédiats :

$$\begin{aligned}
\frac{\partial A}{\partial a}\frac{\partial B}{\partial x} &= \left[-3xy\sin a + xy^2\sin(b-a)\right]\left[2x\cos(2a+3b) + 2y\sin(3a-2b)\right] \\
&= -6x^2y\sin a\cos(2a+3b) - 6xy^2\sin a\sin(3a-2b) \\
&\quad +2x^2y^2\cos(b-a)\cos(2a+3b) + 2xy\cos(b-a)\sin(3a-2b) \\
&= -x^2y\left[\sin(3a+3b) + \sin(-a-3b)\right] - 3xy^2\left[\cos(-2a+2b) - \cos(4a-2b)\right] \\
&\quad +x^2y^2\left[\cos(-3a-2b) + \cos(a+4b)\right] + xy^3\left[\sin(2a-b) - \sin(-4a+3b)\right] \\
-\frac{\partial A}{\partial x}\frac{\partial B}{\partial a} &= -\left[6x^2 + 3y\cos a - y^2\sin(b-a)\right]\left[-2x^2\sin(2a+3b) + 6xy\cos(3a-2b)\right] \\
&= 12x^4\sin(2a+3b) - 36x^3y\cos(3a-2b) + 6x^2y\cos a\sin(2a+3b) \\
&\quad -18xy^2\cos a\cos(3a-2b) - 2x^2y^2\sin(b-a)\sin(2a+3b) + 6xy^3\sin(b-a)\cos(3a-2b) \\
&= 12x^4\sin(2a+3b) - 36x^3y\cos(3a-2b) + 3x^2y\left[\sin(3a+3b) - \sin(-a-3b)\right] \\
&\quad -9xy^2\left[\cos(-2a+2b) + \cos(4a-2b)\right] - x^2y^2\left[\cos(-3a-2b) - \cos(a+4b)\right] \\
&\quad +3xy^3\left[\sin(2a-b) + \sin(-4a+3b)\right] \\
\frac{\partial A}{\partial b}\frac{\partial B}{\partial y} &= \left[-xy^2\cos(b-a)\right]\left[2x\sin(3a-2b)\right] \\
&= -2x^2y^2\cos(b-a)\sin(3a-2b) \\
&= -x^2y^2\left[\sin(2a-b) - \sin(-4a+3b)\right] \\
-\frac{\partial A}{\partial y}\frac{\partial B}{\partial b} &= -\left[3x\cos a - 2xy\sin(b-a)\right]\left[-3x^2\sin(2a+3b) - 4xy\cos(3a-2b)\right] \\
&= 9x^3\cos a\sin(2a+3b) + 12x^2y\cos a\cos(3a-2b) - 6x^3y\sin(b-a)\sin(2a+3b) \\
&\quad -8x^2y^2\sin(b-a)\cos(3a-2b) \\
&= 4.5x^3\left[\sin(3a+3b) - \sin(-a-3b)\right] + 6x^2y\left[\cos(-2a+2b) + \cos(4a-2b)\right] \\
&\quad -3x^3y\left[\cos(-3a-2b) - \cos(a+4b)\right] - 4x^2y^2\left[\sin(2a-b) + \sin(-4a+3b)\right]
\end{aligned}$$

En sommant les produits précédents, nous obtenons :

$$\begin{aligned}
\{A,B\} &= 4.5x^3\left[\sin(3a+3b) - \sin(-a-3b)\right] \\
&\quad -6x^2y\sin(-a-3b) + 6x^2y\cos(-2a+2b) + 6x^2y\cos(4a-2b) \\
&\quad -12xy^2\cos(-2a+2b) - 6xy^2\cos(4a-2b) \\
&\quad +12x^4\sin(2a+3b) \\
&\quad -36x^3y\cos(3a-2b) - 3x^3y\cos(-3a-2b) + 3x^3y\cos(a+4b) \\
&\quad +4xy^3\sin(2a-b) + 2xy^3\sin(-4a+3b) \\
&\quad +2x^2y^2\cos(a+4b) - 5x^2y^2\sin(2a-b) - 3x^2y^2\sin(-4a+3b)
\end{aligned}$$

Le lecteur peut ensuite vérifier que les résultats obtenus via l'utilisation de la sous-routine `POISSON` concordent avec ceux que nous avons obtenus à la main, comme le montre le fichier de sortie ci-dessous.

```
          SERIES     poissonbrackets         1
           NUMBER OF TERMS :     16


         a   b        x   y              COEF
     sin(   1   3 ) (   2   1)    0.6000000000000000D+01
     sin(   1   3 ) (   3   0)    0.4500000000000000D+01
     cos(   1   4 ) (   2   2)    0.2000000000000000D+01
     cos(   1   4 ) (   3   1)    0.3000000000000000D+01
     sin(   2   3 ) (   4   0)    0.1200000000000000D+02
     sin(   2  -1 ) (   1   3)    0.4000000000000000D+01
     sin(   2  -1 ) (   2   2)   -0.5000000000000000D+01
     cos(   2  -2 ) (   1   2)   -0.1200000000000000D+02
     cos(   2  -2 ) (   2   1)    0.6000000000000000D+01
     cos(   3   2 ) (   3   1)   -0.3000000000000000D+01
     sin(   3   3 ) (   3   0)    0.4500000000000000D+01
     cos(   3  -2 ) (   3   1)   -0.3600000000000000D+02
     cos(   4  -2 ) (   1   2)   -0.6000000000000000D+01
     cos(   4  -2 ) (   2   1)    0.6000000000000000D+01
     sin(   4  -3 ) (   1   3)   -0.2000000000000000D+01
     sin(   4  -3 ) (   2   2)    0.3000000000000000D+01
```

Nous avons également implémenté la version vectorielle de cette sous-routine, et faisons référence au Chapitre 3 (plus particulièrement à la section 3.3) pour de plus amples informations.

```
!*********************************************************************!
!subroutine V_POISSON(serA,serB,res,order)                            !
!The subroutine V_POISSON is the same as POISSON for vectorial operations    !
!                                                                     !
!*********************************************************************!
SUBROUTINE V_POISSON(serA,serB,res,order)
  use MSTABLES
  implicit none
  integer, INTENT(IN)::order
  integer, INTENT(IN)::serA(0:order),serB(0:order)
  integer, INTENT(OUT)::res(0:order)
  integer:: Aq(0:MSordmax)=0,Ap(0:MSordmax)=0,Bq(0:MSordmax)=0,Bp(0:MSordmax)=0
  integer::i,k



  ! validation of "order"
  if(order.lt.0.or.order.gt.MSordmax) then
     MSmessage='error in V_POISSON : the order (1) is out of bound'
     MSintmess(1)=order
```

```
     call ERROR
  endif

  !Check if nvt=nvp
  if(MSnvt.ne.MSnvp) then
     MSmessage='error in V_POISSON : argument (1) is not equal &
          & to (2)'
     MSintmess=0
     MSintmess(1)=MSnvt
     MSintmess(2)=MSnvp
     call ERROR
  endif



  do i=1,MSnvt  !Sum over the number of variables
     do k=0,order !Calculate all derivative up to order
        call PDERT(serA(k),Aq(k),MSnamevt(i))!Calculate dA/dq
        if (k>0) then !In the cas of polynomial derivatives, decrease order by 1
           call PDERP(serB(k),Bp(k-1),MSnamevp(i)) !Calculate dB/dp
           call PDERP(serA(k),Ap(k-1),MSnamevp(i)) !Calculate dA/dp
        end if
        call PDERT(serB(k),Bq(k),MSnamevt(i)) !Calculate dB/dq
     end do


     call V_PROD(Aq,Bp,res,1.D0,order) !Make product and add to res
     call V_ERASE(Aq,order) !Erase all temporary series before continue
     Call V_ERASE(Bp,order)
     call V_PROD(Ap,Bq,res,-1.D0,order) !Make product and substract from res
     call V_ERASE(Ap,order)
     call V_ERASE(Bq,order)

  end do

END SUBROUTINE V_POISSON
```

Nous avons validé cette sous-routine en appliquant la version classique et la version vectorielle à deux séries et en comparant les résultats. Il s'avère que les résultats concordent à la précision machine près. Ces résultats peuvent être consultés à l'Annexe D.

4.2.4 Les intégrateurs numériques

Comme nous l'avons vu précédemment, il était nécessaire d'implémenter les équations hamiltoniennes. De même, il est nécessaire d'implémenter un moyen de les résoudre, à savoir des intégrateurs numériques.

Dans un cadre général, l'intégration numérique est une méthode itérative permettant de résoudre l'équation différentielle (ou plus précisément le problème de Cauchy) suivante :

$$y'(t) = f(t, y(t)), \qquad y(t_0) = y_0 \tag{4.5}$$

où t représente le temps et y_0 est la condition initiale.

L'intégrateur numérique le plus commun est certainement celui de Runge-Kutta d'ordre 4 (abbrégé RK4). Une fois choisit un pas d'intégration h, il faut calculer itérativement

$$\begin{aligned} y_{n+1} &= y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\ t_{n+1} &= t_n + h \end{aligned}$$

où

$$\begin{aligned} k_1 &= f(t_n, y_n) \\ k_2 &= f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1) \\ k_3 &= f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2) \\ k_4 &= f(t_n + h, y_n + hk_3) \end{aligned}$$

Cette méthode est d'ordre 4, ce qui signifie que l'erreur par itération est en ordre $o(h^5)$ avec une erreur totale cummulée en ordre $o(h^4)$, ce qui permet généralement d'avoir une bonne approximation de la solution.

Nous avons implémenté cette méthode dans le cadre particulier de la résolution des équations hamiltoniennes. En effet, celle-ci peut être utilisée pour un problème multidimensionnel en effectuant simplement la méthode sur chaque composante.

Dans le code ci-dessous, nous supposons connaître le vecteur des conditions initiales, noté `y`, de dimension `nvar`, l'intervalle d'intégration donné par `a` et `b` ainsi que le pas d'intégration `h`. Les équations à intégrer sont supposées être stockées dans `nvar` séries indentifiées par le vecteur `name`. En sortie, la sous-routine écrit les résultats de l'intégration dans un fichier nommé `RK4.dat`. La valeur des conditions initiales `y` est modifiée en sortie. Le code présenté se décompose en plusieurs étapes :

1. Déclaration des variables et initialisation du temps
2. Écriture des données initiales dans le fichier de sortie
3. Dans un boucle sur le temps
   - Calcul des coefficients k_i pour $i = 1, 2, 3, 4$.
   - Calcul de l'itération
   - Écriture de l'itération dans le fichier de sortie

```
!=========================================================================!
! Integrator Runge Kutta of order 4 for hamiltonian equations             !
!                                                                         !
! IN                                                                      !
!     y     : initial condition                                           !
!     a,b   : interval of integration                                     !
!     h     : step of integration                                         !
!     name  : the series corresponding to the hamiltonian equation,       !
!             which we have to integrate                                  !
!                                                                         !
! OUT                                                                     !
!     y   : the result of the integration, stored in a file RK4.dat       !
!                                                                         !
!=========================================================================!
```

```
SUBROUTINE RK4(y,a,b,h,name)
  use MSTABLES
  implicit none
  double precision,dimension(MSnvar),INTENT(INOUT)::y
  double precision,dimension(MSnvar)::k1,k2,k3,k4,tmp
  double precision::t
  double precision,INTENT(IN)::a,b,h
  INTEGER, dimension(MSnvar), INTENT(IN)::name
  integer::i

  !Initializing the time
  t=a

  !Writing in file, what you need
  open(unit=10,file='RK4.dat',status='replace')
  write(10,'(e15.5)') t
  do i=1,MSnvar
     write(10,'(e25.16)') y(i)
  end do

  !Integration by RK4 method
  DO WHILE (t<b)
     do i=1,MSnvar
        CALL EVALSER(k1(i),y,name(i))
     end do
     tmp=y+h/2*k1
     do i=1,MSnvar
        CALL EVALSER(k2(i),tmp,name(i))
     end do
     tmp=y+h/2*k2
     do i=1,MSnvar
        CALL EVALSER(k3(i),tmp,name(i))
     end do
     tmp=y+h*k3
     do i=1,MSnvar
        CALL EVALSER(k4(i),tmp,name(i))
     end do
     !Updating y
     y=y+h/6*(k1+2*k2+2*k3+k4)

     t=t+h !Updating time

     !Writing in file, what you need
     write(10,'(e15.5)') t
     do i=1,MSnvar
        write(10,'(e25.16)') y(i)
     end do

  END DO
  close(unit=10)
END SUBROUTINE RK4
```

Cette sous-routine est basée essentiellement sur l'utilisation de `EVALSER` et `HAMILTON`. En effet, `EVALSER`

est nécessaire pour calculer les coefficients k_i, et l'argument de sortie `deriv` de la sous-routine `HAMILTON` correspond au format de l'argument d'entrée `name` de la méthode RK4.

L'utilisateur qui souhaiterait intégrer numériquement autre chose que les équations hamiltoniennes (ou seulement une partie d'entre-elles) peut se servir de cette sous-routine comme base pour un intégrateur personnalisé.

De plus, comme nous le verrons au chapitre suivant, il se peut que suite à un changement de variable, l'utilisateur soit intéressé par une valeur modifiée de la sortie `y`. Libre à lui de modifier le code afin de s'adapter au problème considéré.

Nous avons également implémenté la version vectorielle de cette sous-routine, nommée `V_RK4` et basée cette fois sur les sous-routines `V_EVALSER` et `V_HAMILTON`. Les commentaires concernant le codes sont identiques aux précédents.

```
!***********************************************************************!
!                                                                       !
!SUBROUTINE V_RK4(y,a,b,h,order,name)                                   !
!The subroutine V_RK4 is the same as RK4 for vectorial operations       !
!                                                                       !
!***********************************************************************!
SUBROUTINE V_RK4(y,a,b,h,order,name)
  use MSTABLES
  implicit none

  double precision,dimension(MSnvar),INTENT(INOUT)::y
  double precision,dimension(MSnvar)::k1,k2,k3,k4,tmp
  double precision::t
  double precision,INTENT(IN)::a,b,h
  INTEGER, INTENT(IN)::order
  INTEGER, dimension(1:MSnvar,0:order), INTENT(IN)::name
  integer::i

  !Initialisation du temps
  t=a

  !Writing in file, what you need
  open(unit=10,file='RK4.dat',status='replace')
  write(10,'(e15.5,e25.16,e25.16,e25.16,e25.16)') t,y(1),y(2),y(3),y(4)

  !Integration by RK4 method
  DO WHILE (t<b)
     do i=1,MSnvar
        CALL V_EVALSER(k1(i),y,name(i,:),order)
     end do
     tmp=y+h/2*k1
     do i=1,MSnvar
        CALL V_EVALSER(k2(i),tmp,name(i,:),order)
     end do
     tmp=y+h/2*k2
     do i=1,MSnvar
        CALL V_EVALSER(k3(i),tmp,name(i,:),order)
     end do
```

```
     tmp=y+h*k3
     do i=1,MSnvar
        CALL V_EVALSER(k4(i),tmp,name(i,:),order)
     end do
     !Mise a jour du point courant!Updating y
     y=y+h/6*(k1+2*k2+2*k3+k4)

     t=t+h !Updating time

     !Writing in file, what you need
     write(10,'(e15.5,e25.16,e25.16,e25.16,e25.16)') t,y(1),y(2),y(3),y(4)
  END DO
  close(unit=10)
END SUBROUTINE V_RK4
```

Chapitre 5

Upsilon-Andromedae : Une utilisation concrète

Le but de ce chapitre sera d'utiliser le MSNam dans le cadre d'un problème concret, concernant le sytème planétaire Upsilon-Andromedae.
Nous partirons donc des données réelles du système afin d'obtenir l'hamiltonien du système, de calculer les équations hamiltoniennes puis de les intégrer numériquement afin de découvrir l'évolution des excentricités et des longitudes du péricentre à long terme.

5.1 Hamiltonien du problème coplanaire

Comme nous l'avons vu précédemment, si l'on considère un problème des trois corps formé par une étoile de masse m_0 et deux planètes de masses m_1 et m_2 et de demi-grands axes a_1 et a_2, nous avions obtenu l'hamiltonien (1.23) limité au second ordre des masses que nous rappelons ci-dessous.

$$\begin{aligned}\mathcal{H} &= -\frac{Gm_0m_1}{2a_1} - \frac{Gm_0m_2}{2a_2} \\ &\quad -\frac{Gm_1m_2}{a_2} \sum_{k,i_l,j_l,l\in\underline{4}} A_{i_l}^{k,j_l} e_1^{|j_1|+2i_1} e_2^{|j_2|+2i_2} \left(\sin\frac{i_1}{2}\right)^{|j_3|+2i_3} \left(\sin\frac{i_2}{2}\right)^{|j_4|+2i_4} \cos\Phi, \end{aligned} \tag{5.1}$$

où $\Phi = (k+j_1+j_3)\lambda_1 - (k+j_2+j_4)\lambda_2 - j_1\varpi_1 + j_2\varpi_2 - j_3\Omega_1 + j_4\Omega_2$ est la combinaison d'angles.
Les indices $(k, i_l, j_l, l \in \underline{4})$ sont des entiers et les coefficients $A_{i_l}^{k,j_l}$ sont réels et dépendent du rapport des demi-grands axes.
Nous allons ici considérer que les orbites des deux planètes sont coplanaires. Nous pouvons donc considérer uniquement la partie indépendante des inclinaisons, afin d'obtenir l'hamiltonien du problème coplanaire sous la forme

$$\mathcal{H} = -\frac{Gm_0m_1}{2a_1} - \frac{Gm_0m_2}{2a_2} - \frac{Gm_1m_2}{a_2} \sum_{k,i_1,i2,j_1,j_2} A_{i_1,i_2}^{k,j_1,j_2} e_1^{|j_1|+2i_1} e_2^{|j_2|+2i_2} \cos\Phi, \tag{5.2}$$

où $\Phi = (k+j_1)\lambda_1 - (k+j_2)\lambda_2 - j_1\varpi_1 + j_2\varpi_2$ est la combinaison d'angles.
Nous avons ensuite décidé de choisir comme variables canoniques les variables de Delaunay modifiées, limitées également au second ordre des masses, à savoir

$$\begin{array}{lll} \lambda_i = & \text{longitude moyenne de } m_i & L_i = \quad m_i\sqrt{Gm_0a_i} \\ p_i = & \text{- longitude moyenne du péricentre de } m_i \ (=-\varpi_i) & P_i = \quad L_i\left[1-\sqrt{1-e_i^2}\right] \end{array}$$

De plus, à la place des excentricités, nous utiliserons les expressions $E_i = \sqrt{\frac{2P_i}{L_i}}$ liées aux variables de Delaunay modifiées. Une étude peut être effectuée pour montrer que pour des excentricités petites à modérées, $E_i \approx e_i$ (cf. [Libert, 2007] par exemple).
On obtient alors un hamiltonien similaire au précédent dans les nouvelles variables, donné par

$$\mathcal{H} = -\frac{Gm_0m_1}{2a_1} - \frac{Gm_0m_2}{2a_2} - \frac{Gm_1m_2}{a_2} \sum_{k,i_1,i2,j_1,j_2} B_{i_1,i_2}^{k,j_1,j_2} E_1^{|j_1|+2i_1} E_2^{|j_2|+2i_2} \cos\Phi, \tag{5.3}$$

où $\Phi = (k+j_1)\lambda_1 - (k+j_2)\lambda_2 - j_1p_1 + j_2p_2$.
Enfin, comme nous allons nous intéresser à l'étude de la dynamique séculaire du problème, nous allons moyenniser l'hamiltonien par rapport aux courtes périodes. Nous avons donc retiré les termes dépendant des longitudes moyennes de planètes. Cela revient à faire une moyennisation au premier ordre des masses. Nous obtenons alors finalement la formulation suivante

$$\mathcal{K} = -\frac{Gm_1m_2}{a_2} \sum_{k,i_1,i_2} C_{i_1,i_2}^{k} E_1^{k+2i_1} E_2^{k+2i_2} \cos k(p_1 - p_2), \tag{5.4}$$

où a_i, E_i et p_i représentent les valeurs moyennées par rapport aux courtes périodes. Dans la suite, nous les considérerons égales aux valeurs initiales, le changement n'étant pas significatif.

5.2 Les données du problème

Outre la série (5.4) évaluée en $\alpha = \frac{a_1}{a_2} = 0.328$ (voir Annexe C), nous disposons des données suivantes, où les notations $M_\odot, M_\jupiter$ et UA correspondent respectives aux masses du Soleil, de Jupiter et à l'unité astronomique :

e_1 = 0.224
e_2 = 0.267
ϖ_1 = 250.8°
ϖ_2 = 269.7°
m_0 = 1.31 $M_\odot$
m_1 = 1.92 $M_\jupiter$
m_2 = 4.13 $M_\jupiter$
a_1 = 0.832 UA
a_2 = 2.53 UA

Nous allons travailler dans les unités suivantes, à savoir l'unité astronomique (UA) pour la distance, la masse solaire ($M_\odot$) pour la masse et l'année pour le temps. Ce changement d'unité implique que la constante de la gravitation G doit être adaptée en conséquence :

$$G = 4\pi^2. \tag{5.5}$$

En effet, ceci découle de l'équation de Kepler reliant la période T au moyen mouvement n :

$$T = \frac{2\pi}{n} = \frac{2\pi}{\sqrt{\frac{GM_\odot}{a^3}}} \tag{5.6}$$

Il nous reste à préciser que le rapport de la masse de Jupiter sur celle du Soleil vaut :

$$\frac{M_\jupiter}{M_\odot} = \frac{1}{1047.355}, \tag{5.7}$$

et donc les valeurs des masses m_i exprimées en masses solaires sont :

$$m_0 = 1.31 \tag{5.8}$$
$$m_1 = 1.92\frac{1}{1047.355} = 0.0018 \tag{5.9}$$
$$m_2 = 4.13\frac{1}{1047.355} = 0.0039 \tag{5.10}$$

5.3 Évolution des éléments orbitaux

Les données initiales sont présentées en termes d'excentricités e_1, e_2 et des longitudes des péricentres ϖ_1, ϖ_2. Cependant, notre hamiltonien est donné en termes de variables E_i, p_i où, pour rappel, sont définis

$$E_i = \sqrt{\frac{2P_i}{L_i}} \quad \text{avec} \quad \begin{aligned} L_i &= m_i\sqrt{Gm_0a_i} \\ P_i &= L_i\left[1-\sqrt{1-e_i^2}\right] \end{aligned} \tag{5.11}$$
$$p_i = -\varpi_i, \quad i = 1, 2. \tag{5.12}$$

Cela signifie donc qu'il faut modifier l'excentricité e_i en la variable E_i via la formule

$$E_i = \sqrt{2P_i/L_i} = \sqrt{2\left[1-\sqrt{1-e_i^2}\right]}.$$

Les variables (E, p) n'étant pas canoniques, elles ne nous permettent donc pas de travailler avec les équations hamiltoniennes classiques. En fait, les variables canoniques sont les variables (P, p) qui donnent donc lieu aux équations hamiltoniennes

$$\dot{P}_i = -\frac{\partial H}{\partial p_i} \tag{5.13}$$
$$\dot{p_i} = \frac{\partial H}{\partial P_i}, \quad i = 1, 2. \tag{5.14}$$

Afin de pouvoir observer l'évolution des éléments orbitaux dans les variables (E, p), il va falloir calculer des pseudo-équations hamiltoniennes en partant de celles ci-dessus. Il suffit donc de déterminer $\dot{E}_i$ et $\dot{p}_i$ en fonction des dérivées que nous sommes en mesure de calculer.

Par la règle de dérivation en chaîne, nous obtenons les résultats suivants :

$$\dot{E}_i = \frac{\partial E_i}{\partial P_i}\frac{\partial P_i}{\partial t} \tag{5.15}$$
$$\begin{aligned} \dot{p}_i &= \frac{\partial H}{\partial P_i} \\ &= \frac{\partial H}{\partial E_i}\frac{\partial E_i}{\partial P_i}, \quad i = 1, 2. \end{aligned} \tag{5.16}$$

Or, dans l'équation (5.15), nous connaissons $\dfrac{\partial P_i}{\partial t}$ qui n'est autre que $\dot{P}_i$. Dans les deux équations, nous pouvons également calculer le terme $\dfrac{\partial E_i}{\partial P_i}$ qui vaut :

$$\frac{\partial E_i}{\partial P_i} = \frac{1}{2\sqrt{\dfrac{2P_i}{L_i}}}\frac{2}{L_i} = \frac{1}{E_iL_i}, \quad i = 1, 2. \tag{5.17}$$

Dans (5.16), nous pouvons calculer $\frac{\partial H}{\partial E_i}$ sans problème. Donc, si nous rassemblons (5.13), (5.15), (5.16) et (5.17), nous obtenons enfin nos pseudo-équations hamiltoniennes :

$$\dot{E}_i = -\frac{1}{E_i L_i}\frac{\partial H}{\partial p_i} \tag{5.18}$$

$$\dot{p}_i = \frac{1}{E_i L_i}\frac{\partial H}{\partial E_i}, \quad i = 1, 2. \tag{5.19}$$

À cause de ces équations, il faudra non seulement dériver l'hamiltonien par rapport aux variables (E_i, p_i) mais il faudra également diviser le résultat par la constante L_i (via `SCALE`) et diviser la série par le monôme E_i (via `XMON`).

5.4 Le code

Le code ci-dessous permet d'effectuer les opérations précédemment décrites. Nous pouvons le résumer en quelques étapes :

1. Déclaration des variables et initialisation du MSNam.
2. Calculs des paramètres.
3. Donnée des conditions initiales d'intégration et transformation de celles-ci dans les variables utilisées.
4. Lecture de la série.
5. Calcul des équations hamitloniennes via `HAMILTON`.
6. Modifications dues à (5.19).
7. Intégration numérique via `RK4`.

```
! ********************************************************************
program uppsi
! ********************************************************************
  use MSTABLES
  use integrateurs
  implicit none
  integer:: HH=0
  integer:: deriv(MSnvar)=0
  integer:: sc,arg(MSnvt)=0,exp(MSnvp)=0,NBTERM
  integer:: i,j,k,index,argin
  double precision:: coef,m0,m1,m2,a1,a2,pi,G,cst,L1,L2
  double precision:: e1,e2,w1,w2,y(MSnvar)
  double precision::a,b,h
  call START_MS

  !parametres
  m0=1.31D0   !Masses des trois corps consideres
  m1=1.92D0/1047.355D0
  m2=4.13D0/1047.355D0
  a1=0.832D0   !Demi-grands axes
  a2=2.53D0
  pi=3.14159265358979323D0
  G=4*pi**2
```

```
  cst=-G*m1*m2/a2
  L1=m1*sqrt(G*m0*a1)
  L2=m2*sqrt(G*m0*a2)

  !conditions initiales
  e1=0.224D0
  e2=0.267D0
  w1=250.8D0
  w2=269.7D0
  y(1)=sqrt(2*(1-sqrt(1-e1**2))) !E1
  y(2)=sqrt(2*(1-sqrt(1-e2**2))) !E2
  y(3)=- w1*pi/180.D0  !w1 en radian
  y(4)=- w2*pi/180.D0  !w2 en radian


  !lecture dans le fichier
  open(unit=1,file='series.dat',status='old')


19 read(1,*,end=20) argin,exp,coef
  sc=0
  arg(1)=-argin
  arg(2)=argin

  coef=coef*cst
  call STORE(HH,sc,arg,exp,coef) !stockage de la serie

  goto 19
20 close(unit=1)


  !calcul des derivees
  call HAMILTON(HH,deriv)

  !Scaling du aux equations pseudo-hamiltoniennes
  do i=1,MSnvar
     if ((i.eq.1).or.(i.eq.3)) then
        call XMON(deriv(i),MSnamevp(1),-1)
        call SCALE(deriv(i),1.d0/L1)
     else
        call XMON(deriv(i),MSnamevp(2),-1)
        call SCALE(deriv(i),1.d0/L2)
     end if
  end do

  !nettoyage de la memoire
  call ERASE(HH)


  !paramatres pour l'integration numerique
  a=0.d0
  b=50000.d0
```

```
  h=10.d0


  !integration via methode RK4
  call RK4(y,a,b,h,deriv)


  !nettoyage de la memoire
  do i=1,MSnvar
     call ERASE(deriv(i))
  end do

end program uppsi
```

Suite à toutes les remarques précédentes concernant les changements de variables que nous avons effectués, nous avons légèrement modifié la sous-routine `RK4` pour les besoins du problème. En effet, nous avons décidé d'inclure le changement de E_i vers e_i ainsi que le changement de signe de p_i vers ϖ_i au sein de la sous-routine. Nous avons donc utilisé la version de `RK4` ci-dessous

```
  SUBROUTINE RK4(y,a,b,h,name)
    use MSTABLES
    implicit none
    double precision,dimension(MSnvar),INTENT(INOUT)::y
    double precision,dimension(MSNvar)::k1,k2,k3,k4,tmp
    double precision::t
    double precision,INTENT(IN)::a,b,h
    INTEGER, dimension(MSnvar), INTENT(IN)::name
    double precision::e1,e2  ! In our particular case
    integer::i

    !Initializing the time
    t=a

    !In our case, we need to modify some variables
    e1=sqrt(1-(1-(y(1)**2)/2)**2)
    e2=sqrt(1-(1-(y(2)**2)/2)**2)

    !Writing in file, what you need
    open(unit=1,file='RK4.dat',status='replace')
    write(1,'(e15.5,e25.16,e25.16,e25.16,e25.16)') t,e1,e2,-y(3),-y(4)

    !Integration by RK4 method
    DO WHILE (t<b)

       !Corps de la sous-routine inchangÃ©

       !In our case, modify the variables
       e1=sqrt(1-(1-(y(1)**2)/2)**2)
       e2=sqrt(1-(1-(y(2)**2)/2)**2)

       !Writing in file, what you need
```

```
      write(1,'(e15.5,e25.16,e25.16,e25.16,e25.16)') t,e1,e2,-y(3),-y(4)
    END DO
    close(unit=1)
  END SUBROUTINE RK4
```

Les résultats de l'intégration sont stockés dans un fichier nommé `RK4.dat`. Il suffit de le charger avec Matlab pour obtenir les figures que nous commenterons à la section suivante.

5.5 Les résultats

Nous avons effectué une intégration numérique des équations hamiltoniennes obtenues précédemment sur 50.000 ans, par période de 10 ans, soit 5000 pas d'intégration avec la méthode de Runge Kutta.

Puisque nous avons fait l'intégration dans les variables (E_i, p_i) et que nous nous intéressions aux variables (e_i, ϖ_i), nous avons pris l'initiative de faire le changement de variables au sein de la sous-routine `RK4` (cf. code à la section 4.4). Nous pouvons comparer ces résultats à ceux obtenus numériquement grâce à l'intégrateur du problème des n-corps SWIFT (problème complet, sans moyennisation).
À la figure 5.1, nous pouvons constater l'évolution des excentricités des deux planètes avec le temps, comparées avec les solutions du problème complet. Cette simple intégration numérique basée sur un système moyennisé donne une courbe à l'allure très proche de la solution donnée par SWIFT. Il est donc clair qu'en première approximation, cette solution nettement plus facile à obtenir est largement acceptable.

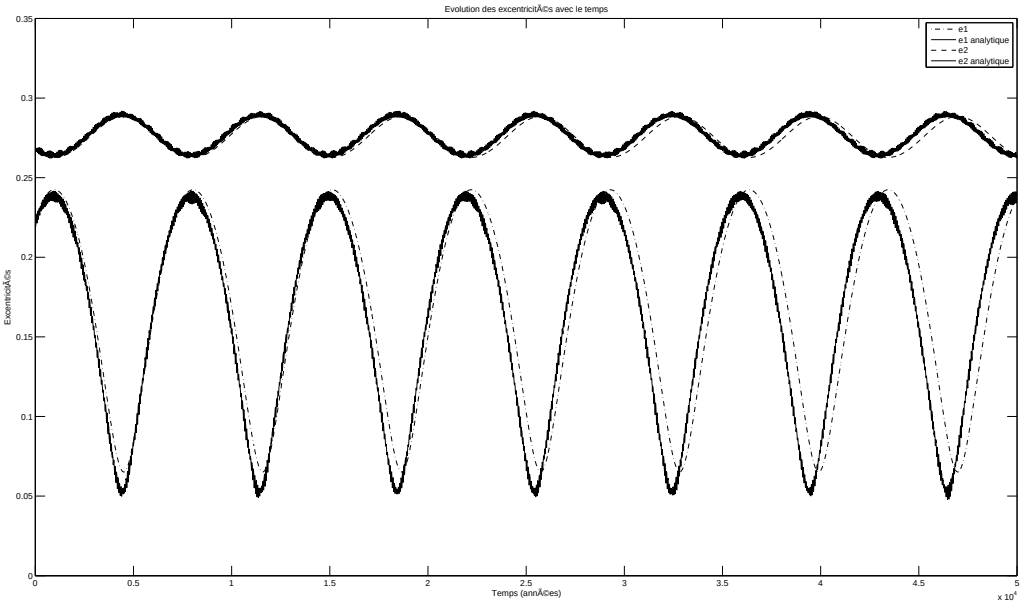


FIGURE 5.1 – Évolution temporelle des excentricités

La Figure 5.2 nous montre le même genre de résultats concernant l'évolution de $\Delta\varpi$, la différence des longitudes du péricentre.

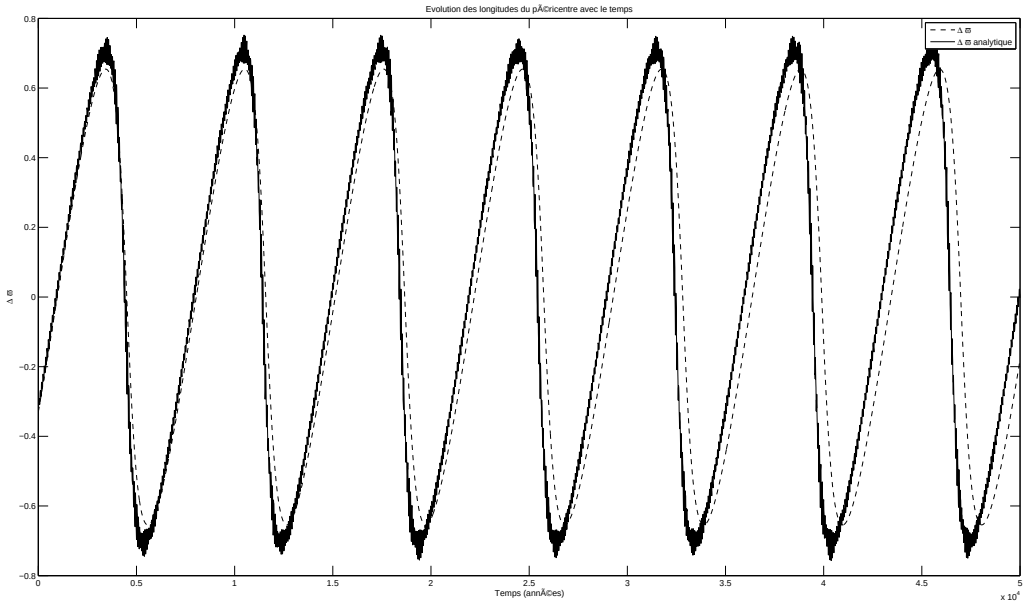


FIGURE 5.2 – Évolution temporelle des longitudes des péricentres

Par la suite, il serait intéressant de tester un système plus complexe (sans moyennisation) avec un intégrateur numérique plus robuste que Runge Kutta afin d'essayer de reproduire plus exactement la solution réelle.

Chapitre 6

Une sous-routine à la loupe : `dev2B_fr`

Le but de ce chapitre est de comprendre en profondeur le fonctionnement d'une sous-routine bien spécifique et relativement complexe : `dev2B_fr`. Pour cela, nous allons développer brièvement une branche de la théorie des perturbations : les transformées de Lie.

Ensuite, nous résolverons deux exercices classiques de mécanique céleste grâce aux transformées de Lie. Il s'agira de développer la distance normalisée ρ et le cosinus de l'anomalie vraie ($\cos f$) en séries en excentricité (c'est-à-dire des séries développées en puissances de l'excentricité e), dépendant uniquement de l'anomalie moyenne M. Nous effectuerons ces calculs jusqu'à l'ordre 3 puis nous les vérifierons et les développerons à un ordre supérieur grâce au MSNam.

Enfin, nous détaillerons le fonctionnement de la sous-routine `dev2B_fr` associée aux développements faits précédemment.

6.1 Théorie des perturbations et transformée de Lie

Dans cette section, nous expliquerons en quoi consiste la théorie des perturbations et décrirons en détails, mais sans démonstration, la méthode de la transformée de Lie. Cette dernière ainsi que l'algorithme qui en découle sera utilisée pour résoudre les problèmes qui nous intéressent.

Le but de ce travail étant plus applicatif que théorique, les notes de cette section ont été fortement inspirées de [Henrard, 2006]. Les résultats seront donnés sans démonstration.

6.1.1 Introduction

La méthode des perturbations s'intéresse aux systèmes différentiels proches de systèmes intégrables du type

$$\dot{x} = L(x) + \epsilon P(x), \qquad x \in \mathbb{R}^n \tag{6.1}$$

où $\dot{x} = L(x)$ est un système intégrable, ϵ un petit paramètre qui peut être soit un paramètre physique, soit un facteur d'échelle, et $P(x)$ une perturbation.
Le but des techniques de perturbation est de construire, sur base de la solution générale du système avec $\epsilon = 0$, des approximations de la solution générale pour $\epsilon \neq 0$ sous forme de développements en série du petit paramètre.
En général, les méthodes de perturbations s'adressent à des systèmes oscillants et les solutions sont observables pour des temps longs.
Nous ne nous attarderons pas plus sur la théorie des perturbations elle-même mais nous allons découvrir en

détail une des méthodes utilisées : la transformée de Lie. Le lecteur intéressé peut se référer à [Henrard, 2006] pour de plus amples informations et exemples concernant la théorie des perturbations.

6.1.2 Transformée de Lie

Nous allons nous intéresser à des transformations des coordonnées $y \in \mathbb{R}^n$ vers les coordonnées $x \in \mathbb{R}^n$, proches de l'identité et analytiques :

$$x = \mathcal{X}(y, \epsilon) = y + \epsilon \mathcal{X}_1(y) + \epsilon^2 \mathcal{X}_2(y) + \dots \tag{6.2}$$

Pour ϵ suffisamment petit, cette transformation est inversible et nous noterons son inverse :

$$y = \mathcal{Y}(x, \epsilon) = x + \epsilon \mathcal{Y}_1(x) + \epsilon^2 \mathcal{Y}_2(x) + \dots \tag{6.3}$$

Nous définirons cette transformation comme la solution d'un système d'équations différentielles

$$\frac{dx}{d\epsilon} = \mathcal{W}(x, \epsilon), \tag{6.4}$$

pour les conditions initiales $x(\epsilon = 0) = y$.
En fait, nous allons considérer la transformation comme un flot qui peut être engendré par un système d'équations différentielles. Localement (pour ϵ suffisamment petit), cela peut toujours se faire. Il suffit, étant donnée la transformation $x = \mathcal{X}(y, \epsilon)$, de prendre pour champ de vecteurs générateur le champ de vecteurs

$$\mathcal{W}(x, \epsilon) = \left[\frac{\partial \mathcal{X}(y, \epsilon)}{\partial \epsilon} \right]_{|y = \mathcal{Y}(x, \epsilon)}. \tag{6.5}$$

Nous remarquons donc que, dans les formules de transformation, c'est le champ de vecteurs $\mathcal{W}(x, \epsilon)$ qui est utile et pas la transformation $\mathcal{X}(x, \epsilon)$ elle-même. C'est pour cela que, dans les algorithmes, nous utiliserons exclusivement $\mathcal{W}$ et non $\mathcal{X}$.

6.1.3 Transformée d'une fonction

La transformée $g(y, \epsilon)$ de toute fonction analytique $f(x, \epsilon)$ par la transformation (6.2) engendrée par le champ de vecteurs générateur (6.4) est donnée par :

$$g(y, \epsilon) = f(\mathcal{X}(y, \epsilon), \epsilon) = \sum_{i \geqslant 0} \frac{\epsilon^i}{i!} \left[D^i f(x, \epsilon) \right]_{|x = y; \epsilon = 0}, \tag{6.6}$$

où l'opérateur D est défini par :

$$Df(x, \epsilon) = \frac{\partial f}{\partial \epsilon} + \left(\frac{\partial f}{\partial x} \right) \bullet \mathcal{W}(x, \epsilon). \tag{6.7}$$

Ceci résulte simplemet du développement de Taylor en ϵ de la fonction $f(\mathcal{X}(y, \epsilon), \epsilon)$ autour de $\epsilon = 0$. Notons que, dans (6.7), la notation $\dfrac{\partial f}{\partial x} \bullet \mathcal{W}$ désigne le produit scalaire du gradient de f et du vecteur $\mathcal{W}$.

6.1.4 Algorithme

Afin d'obtenir une version implémentable, nous pouvons faire le parallèle avec la formule (6.6) pour obtenir un algorithme de calcul assez simple.

Considérons la fonction analytique

$$f(x,\epsilon) = \sum_{n\geqslant 0} \frac{\epsilon^n}{n!} f_n^{(0)}(x) \tag{6.8}$$

et le champ de vecteurs générateur

$$\mathcal{W}(x,\epsilon) = \sum_{n\geqslant 0} \frac{\epsilon^n}{n!} \mathcal{W}_{n+1}(x). \tag{6.9}$$

On construit successivement les fonctions intermédiaires $f_n^{(i)}(x)$ du développement des dérivées

$$\frac{d^i}{d\epsilon^i} f(\mathcal{X}(y,\epsilon),\epsilon) = \sum_{n\geqslant 0} \frac{\epsilon^n}{n!} f_n^{(i)}(x) \tag{6.10}$$

par la formule de récurrence :

$$f_n^{(i)} = f_{n+1}^{(i-1)} + \sum_{j=0}^{n} C_n^j L_{j+1} f_{n-j}^{(i-1)} \tag{6.11}$$

où l'opérateur différentiel L_m est la dérivée de Lie dans la direction $\mathcal{W}_m$, c'est-à-dire

$$L_m h = \frac{\partial h}{\partial x} \bullet \mathcal{W}_m. \tag{6.12}$$

La transformée $g(y,\epsilon)$ de la fonction $f(x,\epsilon)$ est alors donnée par

$$g(y,\epsilon) = \sum_{n\geqslant 0} \frac{\epsilon^n}{n!} \left[f_0^{(n)}(x) \right]_{|x=y}. \tag{6.13}$$

La récurrence est plus simple à imaginer si l'on considère les fonction intermédiaires $f_j^{(i)}$ arrangées comme ci-dessous, en un triangle appelé triangle de Lie :

$$\begin{array}{cccccc}
f_0^{(0)} & & & & & \\
f_1^{(0)} & f_0^{(1)} & & & & \\
f_2^{(0)} & f_1^{(1)} & f_0^{(2)} & & & \\
f_3^{(0)} & f_2^{(1)} & f_1^{(2)} & f_0^{(3)} & & \\
\vdots & & & & \ddots &
\end{array}$$

On voit que la formule de transformation (6.13) nécessite de calculer les éléments de la « diagonale ». Ceux-ci étant calculés via la formule de récurrence (6.11) au moyen de toutes les fonctions de la colonne précédente situées au-dessus ou à la même hauteur et seulement ces fonctions-là.

Le lecteur intéressé par la démonstration de la formule (6.11) peut se référer à [Henrard, 2006]. Il suffit simplement d'effectuer la dérivation de l'équation (6.10) puis de faire une identification terme à terme.

6.2 Les développements « à la main »

Comme nous l'avons dit précédemment, le but de ce chapitre sera, entre autres, d'effectuer deux développements en série en excentricité, relatifs au problème des deux corps. Commençons donc cette section par quelques rappels de formules relatives au problème des deux corps.

Définissons $\rho = \frac{r}{a}$ la distance normalisée entre les deux corps. Nous connaissons les formules suivantes :

$$\rho \cos f = \cos E - e, \qquad \rho \sin f = \sqrt{1-e^2}\sin E \tag{6.14}$$

$$\rho = 1 - e\cos E \tag{6.15}$$

$$\rho = \frac{1-e^2}{1+e\cos f} \tag{6.16}$$

$$M = E - e\sin E \tag{6.17}$$

où f est l'anomalie vraie, E l'anomalie excentrique, M l'anomalie moyenne et e l'excentricité.
Nous supposerons que ρ, f et E sont des fonctions de e et M, considérées comme variables indépendantes.
Les deux développements que nous effectuerons sont les suivants :

- La distance normalisée ρ en fonction de l'anomalie moyenne M.
- Le cosinus de l'anomalie vraie, $\cos f$, en fonction de M.

Nous effectuerons ces développements via la transformée de Lie jusqu'à l'ordre 3 en excentricité, c'est-à-dire que nous considérerons e comme le petit paramètre ϵ. L'excentricité étant un paramètre généralement petit, c'est pour cela que les développements en excentricité sont généralement fréquents. Par la suite, nous vérifierons et continuerons ces développements de manière numérique.

6.2.1 Les fonctions génératrices

Comme nous l'avons vu à la section précédente, pour effectuer une transformée de Lie, il nous faut une fonction génératrice. Au vu de la définition (6.4), nous allons définir trois fonctions génératrices (pour E, ρ et f), mais nous n'utiliserons que les deux dernières.

Pour ce faire, considérons l'équation de Kepler (6.17) comme un changement de variables de E vers M, correspondant à la forme (6.3).
Afin d'obtenir la fonction génératrice, en respectant la formule (6.4), il suffit de dériver (6.17) par rapport à e, nous obtenons

$$\begin{aligned} 0 &= \frac{\partial E}{\partial e} - \sin E - e\cos E\frac{\partial E}{\partial e} \\ &= -\sin E + (1 - e\cos E)\frac{\partial E}{\partial e} \\ &= -\sin E + \rho\frac{\partial E}{\partial e} \end{aligned} \tag{6.18}$$

On peut alors extraire $\dfrac{\partial E}{\partial e}$ et utiliser (6.14) afin d'obtenir

$$\frac{\partial E}{\partial e} = \frac{\sin f}{\sqrt{1-e^2}} \tag{6.19}$$

La seconde génératrice est donnée en dérivant (6.15) par rapport à e.

$$\frac{\partial \rho}{\partial e} = -\cos E + e\sin E\frac{\partial E}{\partial e} \tag{6.20}$$

Nous utilisons ensuite (6.14) pour exprimer $\sin E$ et $\cos E$ en fonction de ρ et f, ainsi que (6.19) afin d'obtenir

$$\begin{aligned}
\frac{\partial \rho}{\partial e} &= -\rho \cos f - e + \frac{e \sin f}{\sqrt{1-e^2}} \frac{\rho \sin f}{\sqrt{1-e^2}} \\
&= \frac{\rho}{1-e^2}[e \sin^2 f - \cos f + e^2 \cos f] - e \\
&= \frac{1}{1+e\cos f}[e - e\cos^2 f - \cos f + e^2 \cos f] - e \qquad \text{par (6.16)} \\
&= \frac{1}{1+e\cos f}[(1+e\cos f)(e - \cos f)] - e \\
&= -\cos f
\end{aligned} \tag{6.21}$$

Enfin, pour obtenir la troisième génératrice, il faut faire quelques manipulations supplémentaires. Pour cela, nous allons d'abord dériver (6.16) par rapport à e en utilisant la règle de dérivée d'un quotient.

$$\begin{aligned}
\frac{\partial \rho}{\partial e} &= \frac{\partial}{\partial e} \frac{1-e^2}{1+e\cos f} \\
&= \frac{-2e(1+e\cos f)}{(1+e\cos f)^2} - \frac{1-e^2}{(1+e\cos f)^2} \frac{\partial}{\partial e}(1+e\cos f) \\
&= \frac{-2e}{1+e\cos f} - \frac{1-e^2}{(1+e\cos f)^2} \left(\cos f - e \sin f \frac{\partial f}{\partial e} \right)
\end{aligned} \tag{6.22}$$

Afin d'en tirer $\dfrac{\partial f}{\partial e}$, nous allons utiliser (6.21). On obtient alors

$$\begin{aligned}
\frac{1-e^2}{(1+e\cos f)^2} \left(e \sin f \frac{\partial f}{\partial e} \right) &= \frac{\partial \rho}{\partial e} + \frac{2e}{1+e\cos f} + \frac{1-e^2}{(1+e\cos f)^2} \cos f \\
&\Updownarrow \\
e \sin f \frac{\partial f}{\partial e} &= \frac{(1+e\cos f)^2}{1-e^2} \frac{\partial \rho}{\partial e} + 2e \frac{1+e\cos f}{1-e^2} + \cos f \\
&= -\frac{(1+e\cos f)^2}{1-e^2} \cos f + 2e \frac{1+e\cos f}{1-e^2} + \cos f \qquad \text{par (6.21)} \\
&= \frac{1}{1-e^2} \left[(-1 - 2e\cos f - e^2 \cos^2 f) \cos f + 2e + 2e^2 \cos f + \cos f - e^2 \cos f \right] \\
&= \frac{1}{1-e^2} \left[-\cos f - 2e\cos^2 f - e^2 \cos^3 f + 2e + 2e^2 \cos f + \cos f - e^2 \cos f \right] \\
&= \frac{e}{1-e^2} \left[-2\cos^2 f - e\cos^3 f + 2 + e\cos f \right] \\
&= \frac{e}{1-e^2} (1 - \cos^2 f)(2 + e\cos f) \\
&= \frac{e \sin^2 f}{1-e^2} (2 + e\cos f).
\end{aligned} \tag{6.23}$$

On obtient alors finalement

$$\frac{\partial f}{\partial e} = \frac{\sin f}{1-e^2} (2 + e\cos f). \tag{6.24}$$

On peut également utiliser la formule de l'angle double du sinus afin d'obtenir une formulation qui ne dépend que de $\sin f$ et de e.

$$\begin{aligned}\frac{\partial f}{\partial e} &= \frac{2\sin f + e\sin f\cos f}{1-e^2} \\ &= \frac{4\sin f + e\sin 2f}{2(1-e^2)}.\end{aligned} \tag{6.25}$$

Cependant, pour utiliser les formules du triangle de Lie, il faut développer les fonctions génératrices en séries. Puisque la fonction génératrice (6.21) ne dépend pas explicitement de e, ce n'est pas nécessaire. Plus précisément, nous la considérons comme contenant uniquement le terme indépendant de e.

Le développement de la fonction génératrice (6.25) en série se fait via le développement du terme $\frac{1}{1-e^2}$ en série. Pour se faire, nous calculer les dérivées successives :

$$\begin{aligned}\left(\frac{1}{1-e^2}\right)' &= \frac{2e}{(1-e^2)^2} \\ \left(\frac{2e}{(1-e^2)^2}\right)' &= \frac{2(1-e^2)^2 + 2e\left[2(1-e^2)(+2e)\right]}{(1-e^2)^4} \\ &= \frac{2(1-e^2)+8e^2}{(1-e^2)^3} \\ &= \frac{6e^2+2}{(1-e^2)^3} \\ \left(\frac{6e^2+2}{(1-e^2)^3}\right)' &= \frac{12e(1-e^2)^3 + (6e^2+2)\left[3(1-e^2)^2(+2e)\right]}{(1-e^2)^6} \\ &= \frac{12e(1-e^2)^3 + (36e^2+12e)(1-e^2)^2}{(1-e^2)^6} \\ &= \frac{12e-12e^3+36e^2+12e}{(1-e^2)^4} \\ &= \frac{12(-e^3+3e^2+2e)}{(1-e^2)^4} \\ \left(\frac{12(-e^3+3e^2+2e)}{(1-e^2)^4}\right)' &= \frac{12(-3^2+6e+2)(1-e^2)^4 + 12(-e^3+3e^2+2e)\left[4(1-e^2)^3(+2e)\right]}{(1-e^2)^8} \\ &= \dots\end{aligned}$$

Nous obtenons alors

$$\begin{aligned}\left[\frac{1}{1-e^2}\right]_{|e=0} &= 1 \\ \left[\left(\frac{1}{1-e^2}\right)^{(1)}\right]_{|e=0} &= 0 \\ \left[\left(\frac{1}{1-e^2}\right)^{(2)}\right]_{|e=0} &= 2 \\ \left[\left(\frac{1}{1-e^2}\right)^{(3)}\right]_{|e=0} &= 0 \\ \left[\left(\frac{1}{1-e^2}\right)^{(4)}\right]_{|e=0} &= 24 = 4!\end{aligned}$$

Le développement en série de la fonction génératrice (6.25) nous donne donc

$$\begin{aligned}\frac{4\sin f + e\sin 2f}{2(1-e^2)} &= \frac{4\sin f + e\sin 2f}{2}\left[1 + 0e + 2\frac{e^2}{2} + 0\frac{e^3}{3!} + 4!\frac{e^4}{4!} + o(e^5)\right] \\ &= 2\sin f + e\frac{\sin 2f}{2} + e^2 2\sin f + e^3\frac{\sin 2f}{2} + e^4 2\sin f + o(e^5) \end{aligned} \tag{6.26}$$

Nous avons arrêté notre développement à l'ordre 5, mais l'on peut démontrer que la génératrice peut s'écrire sous la forme

$$\frac{4\sin f + e\sin 2f}{2(1-e^2)} = 2\sin f\sum_{k=0}^{\infty} e^{2k} + \frac{1}{2}\sin 2f\sum_{k=0}^{\infty} e^{2k+1} \tag{6.27}$$

6.2.2 Développement de $\cos f$ en fonction de M

Grâce au développement en série de la fonction génératrice obtenu en (6.27), nous allons pouvoir effectuer les développements souhaités. Afin de respecter notre algorithme, nous devons écrire la génératrice sous la forme (6.9), c'est-à-dire avec

$$\begin{aligned}\mathcal{W}_1 &= 2\sin f \\ \mathcal{W}_2 &= \frac{\sin 2f}{2} \\ \mathcal{W}_3 &= 4\sin f \\ \mathcal{W}_n &= 0 \quad \text{pour } n \geqslant 4,\end{aligned}$$

car nous arrêtons le développement à l'ordre 3 en e.
La fonction que nous voulons transformer est $f(f,e) = \cos f$, que nous devons réécrire sous la forme (6.8) comme

$$\begin{aligned}f_0^{(0)} &= \cos f \\ f_n^{(0)} &= 0 \quad \text{pour } n \geqslant 1\end{aligned} \tag{6.28}$$

On applique ensuite l'agorithme (6.11) afin de calculer successivement tous les termes du développement. Les calculs font généralement appel à des formules de trigonométrie classiques, ainsi qu'aux formules d'angles multiples.

$$\begin{aligned}f_0^{(1)} &= f_1^{(0)} + C_0^0 L_1 f_0^{(0)} = 0 + \frac{\partial\cos f}{\partial f}\mathcal{W}_1 \\ &= -2\sin^2 f = \cos 2f - 1 \\ f_1^{(1)} &= f_2^{(0)} + C_1^0 L_1 f_1^{(0)} + C_1^1 L_2 f_0^{(0)} \\ &= 0 + 0 + \frac{\partial\cos f}{\partial f}\mathcal{W}_2 \\ &= -\sin f\frac{\sin 2f}{2} \\ &= -\sin f\sin f\cos f \\ &= -\sin^2 f\cos f \\ &= -(1-\cos^2 f)\cos f \\ &= -\cos f + \cos^3 f\end{aligned} \tag{6.29}$$

$$
\begin{aligned}
f_0^{(2)} &= f_1^{(1)} + C_0^0 L_1 f_0^{(1)} \\
&= -\cos f + \cos^3 f + \frac{\partial(\cos 2f - 1)}{\partial f} 2 \sin f \\
&= -\cos f + \cos^3 f - 4 \sin 2f \sin f \\
&= -\cos f + \cos^3 f - 8 \sin^2 f \cos f \\
&= -\cos f + \cos^3 f - 8(1 - \cos^2 f) \cos f \\
&= -9 \cos f + 9 \cos^3 f \\
&= -9 \cos f + \frac{9}{4}(\cos 3f + 3 \cos f) \\
&= \frac{9}{4}(\cos 3f - \cos f). \qquad (6.30)
\end{aligned}
$$

$$
\begin{aligned}
f_2^{(1)} &= f_3^{(0)} + C_2^0 L_1 f_2^{(0)} + C_2^1 L_2 f_1^{(0)} + C_2^2 L_3 f_0^{(0)} \\
&= 0 + 0 + 0 + \frac{\partial \cos f}{\partial f} 4 \sin f = -4 \sin^2 f \\
f_1^{(2)} &= f_2^{(1)} + C_1^0 L_1 f_1^{(1)} + C_1^1 L_2 f_0^{(1)} \\
&= -4 \sin^2 f + \frac{\partial(-\cos f + \cos^3 f)}{\partial f} 2 \sin f + \frac{\partial(\cos 2f - 1)}{\partial f} \frac{\sin 2f}{2} \\
&= -4 \sin^2 f + 2 \sin^2 f - 6 \cos^2 f \sin^2 f - (\sin 2f)^2 \\
f_0^{(3)} &= f_1^{(2)} + C_0^0 L_1 f_0^{(2)} \\
&= -2 \sin^2 f - 6 \cos^2 f \sin^2 f - 4 \sin^2 f \cos^2 f + \frac{9}{4} \frac{\partial(\cos 3f - \cos f)}{\partial f} 2 \sin f \\
&= -2(1 - \cos^2 f) - 10 \cos^2 f (1 - \cos^2 f) + \frac{9}{2} \sin f (-3 \sin 3f + \sin f) \\
&= -2 - 8 \cos^2 f + 10 \cos^4 f + \frac{9}{2}(1 - \cos^2 f) - \frac{27}{2} \sin f \sin 3f \\
&= \frac{5}{2} - \frac{25}{2} \cos^2 f + 10(\frac{1}{8} \cos 4f + \cos^2 f - \frac{1}{8}) - \frac{27}{4}(\cos 2f - \cos 4f) \\
&= \frac{5}{2} - \frac{25}{4}(\cos 2f + 1) + \frac{5}{4} \cos 4f + \frac{10}{2}(\cos 2f + 1) - \frac{5}{4} - \frac{27}{4} \cos 2f + \frac{27}{4} \cos 4f \\
&= \frac{10 - 25 + 20 - 5}{4} + \frac{-25 + 20 - 27}{4} \cos 2f + \frac{5 + 27}{4} \cos 4f \\
&= 8(\cos 4f - \cos 2f) \qquad (6.31)
\end{aligned}
$$

Pour obtenir le résultat final, il suffit d'utiliser (6.28), (6.29), (6.30) et (6.31) en remplaçant f par M et d'en faire un développement comme indiqué dans (6.13). Cela donne une expression de $g(M, e) = \cos f$ en fonction de e et M :

$$
\cos f = \cos M + e(\cos 2M - 1) + \frac{9e^2}{8}(\cos 3M - \cos M) + \frac{4e^3}{3}(\cos 4M - \cos 2M) + o(e^4) \qquad (6.32)
$$

6.2.3 Développement de ρ en fonction de M

Le cas du développement de ρ est plus complexe. En effet, ρ est cette fois considéré comme une variable en soi et non plus commme une fonction de e et E comme précédemment. De ce fait, ρ ne doit plus être développé en série. On considère simplement la fonction $f(\rho, e) = \rho$, le reste étant nul, c'est-à-dire

$$\begin{aligned} f_0^{(0)} &= \rho \\ f_n^{(0)} &= 0 \quad \text{pour } n \geqslant 1 \end{aligned} \tag{6.33}$$

De plus, nous allons devoir utiliser les deux génératrices (6.21) et (6.25) au lieu d'une seule, à chaque étape du triangle de Lie. Leur écriture sous forme développée est :

$$\mathcal{WR} = -\cos f \quad \text{avec}$$

$$\begin{aligned} \mathcal{WR}_1 &= -\cos f \\ \mathcal{WR}_n &= 0 \quad \text{pour } n \geqslant 1 \end{aligned}$$

$$\mathcal{WF} = 2\sin f + e\frac{\sin 2f}{2} + \frac{e^2}{2}4\sin f + o(e^4) \quad \text{avec}$$

$$\begin{aligned} \mathcal{WF}_1 &= 2\sin f \\ \mathcal{WF}_2 &= \frac{\sin 2f}{2} \\ \mathcal{WF}_3 &= 4\sin f \\ \mathcal{WF}_n &= 0 \quad \text{pour } n \geqslant 4, \end{aligned}$$

car une fois de plus nous arrêtons le développement à l'ordre 3 en e.
Puisque nous utiliserons les deux génératrices de manière coordonnée, cela signifie que l'opérateur de dérivation de Lie devient

$$L_m h = \frac{\partial h}{\partial \rho}\mathcal{WR}_m + \frac{\partial h}{\partial f}\mathcal{WF}_m. \tag{6.34}$$

Effectuons maintenant le calcul du triangle de Lie

$$\begin{aligned} f_0^{(1)} &= f_1^{(0)} + C_0^0 L_1 f_0^{(0)} \\ &= 0 + \frac{\partial \rho}{\partial \rho}(-\cos f) + \frac{\partial \rho}{\partial f}2\sin f \\ &= -\cos f + 0 = -\cos f \\ f_1^{(1)} &= f_2^{(0)} + C_1^0 L_1 f_1^{(0)} + C_1^1 L_2 f_0^{(0)} \\ &= 0 + 0 + \frac{\partial \rho}{\partial \rho}0 + \frac{\partial \rho}{\partial f}2\sin f = 0 \end{aligned} \tag{6.35}$$

$$
\begin{aligned}
f_0^{(2)} &= f_1^{(1)} + C_0^0 L_1 f_0^{(1)} \\
&= 0 + \frac{\partial(-\cos f)}{\partial \rho}(-\cos f) + \frac{\partial(-\cos f)}{\partial f} 2\sin f \\
&= 2\sin^2 f = 1 - \cos 2f \qquad (6.36) \\
f_2^{(1)} &= f_3^{(0)} + C_2^0 L_1 f_2^{(0)} + C_2^1 L_2 f_1^{(0)} + C_2^2 L_3 f_0^{(0)} \\
&= 0 + 0 + 0 + \frac{\partial \rho}{\partial \rho} 0 + \frac{\partial \rho}{\partial f} 4\sin f \\
&= 0 \\
f_1^{(2)} &= f_2^{(1)} + C_1^0 L_1 f_1^{(1)} + C_1^1 L_2 f_0^{(1)} \\
&= 0 + 0 \frac{\partial(-\cos f)}{\partial \rho} 0 + \frac{\partial(-\cos f)}{\partial f} \frac{\sin 2f}{2} \\
&= \frac{1}{2} \sin f \sin 2f \\
&= \sin^2 f \cos f = (1 - \cos^2 f)\cos f \\
&= \cos f - \cos^3 f \\
f_0^{(3)} &= f_1^{(2)} + C_0^0 L_1 f_0^{(2)} \\
&= \cos f - \cos^3 f + \frac{\partial(1 - \cos 2f)}{\partial \rho}(-\cos f) + \frac{\partial(1 - \cos 2f)}{\partial f} 2\sin f \\
&= \cos f - \cos^3 f + 0 + 4\sin 2f \sin f \\
&= \cos f - \cos^3 f + 8\sin^2 f \cos f \\
&= \cos f - \cos^3 f + 8\cos f - 8\cos^3 f \\
&= 9\cos f - 9\cos^3 f \\
&= 9\cos f - \frac{9}{4}\cos 3f - \frac{27}{4}\cos f \\
&= \frac{9}{4}(\cos f - \cos 3f) \qquad (6.37)
\end{aligned}
$$

Pour obtenir le résultat final, il suffit d'utiliser (6.33), (6.35), (6.36) et (6.37) en remplaçant f par M et d'en faire un développement comme indiqué dans (6.13). Cela donne une expression de $g(M, e) = \rho$ en fonction de e et M :

$$\rho = 1 - e\cos M + \frac{e^2}{2}(1 - \cos 2M) + \frac{3e^3}{8}(\cos M - \cos 3M) + o(e^4) \qquad (6.38)$$

6.3 Vérification numérique

Afin d'effectuer une vérification et un prolongement numérique des résultats obtenus à la section précédente, nous allons utiliser une routine bien spécifique du MSNam, nommée `dev2B_FR`. Elle permet, comme son nom l'indique, d'effectuer les développements de f et r dans le cadre du problème des deux corps (2-Body). Plus précisément, comme nous l'avons vu au Chapitre 2, le descriptif de la sous-routine est le suivant :

`DEV2B_FR(A,order)` : Présuppose que deux variables polynomiales sont identifiées respectivement par ' r' et ' e' et une variable trigonométrique par ' f'. L'argument `A` en entrée est un vecteur `A(0:order)`, où `A(k)`

est la composante d'ordre `k` dans le développement en puissances de ' e'. Dans le cadre du problème des deux corps, ' r' représente la distance normalisée r/a, ' e' l'excentricité et ' f' l'anomalie vraie.
La sous-routine remplace la fonction `A` par son développement en termes d'excentricité et d'anomalie moyenne. Le résultat est indépendant de la variable ' r' (c'est-à-dire que l'on fixe tous les exposants de ' r' à zéro). Dans la sortie `A`, la variable ' f' désigne l'anomalie moyenne.

Commençons par la vérification et le prolongement des résultats obtenus.

Pour ce faire, nous allons considérer deux séries :

1. $\cos f$
2. ρ

Afin de simplifier le problème, nous ne définirons que 3 variables dans les paramètres du MSNam, à savoir celles qui nous intéressent : ρ, e et f. La routine est plus générale et permet d'effectuer le développement de ces variables dans une série en contenant d'autres. Cependant, nous ne nous intéressons qu'aux développements effectués précédemments.

Le développement de $\cos f$ jusqu'à l'ordre 8 en excentricité donne les résultats ci-dessous :

```
      SERIES     Serie cos f dev         0
       NUMBER OF TERMS :      1


         f       r   e            COEF
cos(   1 ) (   0   0)    0.1000000000000000D+01



      SERIES     Serie cos f dev         1
       NUMBER OF TERMS :      2


         f       r   e            COEF
cos(   0 ) (   0   1)   -0.1000000000000000D+01
cos(   2 ) (   0   1)    0.1000000000000000D+01



      SERIES     Serie cos f dev         2
       NUMBER OF TERMS :      2


         f       r   e            COEF
cos(   1 ) (   0   2)   -0.1125000000000000D+01
cos(   3 ) (   0   2)    0.1125000000000000D+01



      SERIES     Serie cos f dev         3
       NUMBER OF TERMS :      2
```

```
       f       r   e            COEF
cos(   2 ) (   0   3)   -0.1333333333333333D+01
cos(   4 ) (   0   3)    0.1333333333333333D+01


     SERIES     Serie cos f dev         4
      NUMBER OF TERMS :      3


       f       r   e            COEF
cos(   1 ) (   0   4)    0.1302083333333333D+00
cos(   3 ) (   0   4)   -0.1757812500000000D+01
cos(   5 ) (   0   4)    0.1627604166666667D+01


     SERIES     Serie cos f dev         5
      NUMBER OF TERMS :      3


       f       r   e            COEF
cos(   2 ) (   0   5)    0.3750000000000000D+00
cos(   4 ) (   0   5)   -0.2400000000000000D+01
cos(   6 ) (   0   5)    0.2025000000000000D+01


     SERIES     Serie cos f dev         6
      NUMBER OF TERMS :      4


       f       r   e            COEF
cos(   1 ) (   0   6)   -0.5316840277777778D-02
cos(   3 ) (   0   6)    0.7751953125000001D+00
cos(   5 ) (   0   6)   -0.3323025173611111D+01
cos(   7 ) (   0   6)    0.2553146701388889D+01


     SERIES     Serie cos f dev         7
      NUMBER OF TERMS :      4


       f       r   e            COEF
cos(   2 ) (   0   7)   -0.4444444444444445D-01
cos(   4 ) (   0   7)    0.1422222222222222D+01
cos(   6 ) (   0   7)   -0.4628571428571429D+01
cos(   8 ) (   0   7)    0.3250793650793651D+01
```

```
      SERIES     Serie cos f dev         8
       NUMBER OF TERMS :      5


         f       r   e            COEF
cos(   1 ) (   0   8)    0.1098632812500000D-03
cos(   3 ) (   0   8)   -0.1601806640625000D+00
cos(   5 ) (   0   8)    0.2452305385044643D+01
cos(   7 ) (   0   8)   -0.6462652587890625D+01
cos(   9 ) (   0   8)    0.4170418003627232D+01
```

Si nous réécrivons les termes du résultat sous forme fractionnelle, nous retrouvons (jusqu'à l'ordre 4) le développement suivant

$$\cos f = \cos M + e(\cos 2M - 1) + \frac{9e^2}{8}(\cos 3M - \cos M) + \frac{4e^3}{3}(\cos 4M - \cos 2M) + e^4\left(\frac{25}{192}\cos M - \frac{225}{128}\cos 3M + \frac{625}{384}\cos 5M\right) + o(e^5) \tag{6.39}$$

Nous remarquons donc que les résultats concordent avec ce que nous avons fait à la main, et que le développement peut être effectué à des ordres bien supérieurs.

De la même manière, le développement de ρ jusqu'à l'ordre 8 en excentricité donne les résultats ci-dessous :

```
      SERIES     Serie r dev         0
       NUMBER OF TERMS :      1


         f       r   e            COEF
cos(   0 ) (   0   0)    0.1000000000000000D+01



      SERIES     Serie r dev         1
       NUMBER OF TERMS :      1


         f       r   e            COEF
cos(   1 ) (   0   1)   -0.1000000000000000D+01



      SERIES     Serie r dev         2
       NUMBER OF TERMS :      2


         f       r   e            COEF
cos(   0 ) (   0   2)    0.5000000000000000D+00
cos(   2 ) (   0   2)   -0.5000000000000000D+00



      SERIES     Serie r dev         3
```

```
      NUMBER OF TERMS :      2


       f       r   e            COEF
cos(   1 ) (   0   3)    0.3750000000000000D+00
cos(   3 ) (   0   3)   -0.3750000000000000D+00



     SERIES     Serie r dev         4
      NUMBER OF TERMS :      2


       f       r   e            COEF
cos(   2 ) (   0   4)    0.3333333333333333D+00
cos(   4 ) (   0   4)   -0.3333333333333333D+00



     SERIES     Serie r dev         5
      NUMBER OF TERMS :      3


       f       r   e            COEF
cos(   1 ) (   0   5)   -0.2604166666666667D-01
cos(   3 ) (   0   5)    0.3515625000000000D+00
cos(   5 ) (   0   5)   -0.3255208333333333D+00



     SERIES     Serie r dev         6
      NUMBER OF TERMS :      3


       f       r   e            COEF
cos(   2 ) (   0   6)   -0.6250000000000000D-01
cos(   4 ) (   0   6)    0.4000000000000000D+00
cos(   6 ) (   0   6)   -0.3375000000000000D+00



     SERIES     Serie r dev         7
      NUMBER OF TERMS :      4


       f       r   e            COEF
cos(   1 ) (   0   7)    0.7595486111111111D-03
cos(   3 ) (   0   7)   -0.1107421875000000D+00
cos(   5 ) (   0   7)    0.4747178819444444D+00
cos(   7 ) (   0   7)   -0.3647352430555555D+00
```

```
          SERIES     Serie r dev         8
           NUMBER OF TERMS :      4


                f       r   e            COEF
      cos(   2 ) (   0   8)    0.5555555555555556D-02
      cos(   4 ) (   0   8)   -0.1777777777777778D+00
      cos(   6 ) (   0   8)    0.5785714285714286D+00
      cos(   8 ) (   0   8)   -0.4063492063492063D+00
```

Sous forme fractionnelle, cela correspond à

$$\rho = 1 - e\cos M + \frac{e^2}{2}(1-\cos 2M) + \frac{3e^3}{8}(\cos M - \cos 3M) + \frac{e^4}{3}(\cos 2M - \cos 4M) + o(e^5) \tag{6.40}$$

Nous voyons donc une fois de plus que les résultats concordent et qu'ils ont été développés plus loin.

6.4 Descriptif de dev2B_fr

Passons à présent au descriptif de la sous-routine `dev2B_fr` du MSNam en elle-même. Dans cette section, nous allons décortiquer étape par étape ce qui est fait au sein de la routine afin de la comprendre en profondeur.

La première partie est constituée des traditionnelles déclarations de variables du MSNam sur lesquelles nous ne nous attarderons pas. Ensuite, l'étape de validation permet de déterminer si l'on peut utiliser la sous-routine. On y vérifie tout d'abord que l'ordre `order` est bien positif et inférieur à l'ordre maximal `ordmax`. On vérifie également l'identifiant de la série à chaque ordre. Pour cela, on contrôle d'une part que l'identifiant est positif (sinon la série serait mal ordonnée) et d'autre part, on s'assure que l'identifiant ne dépasse pas le nombre maximal de séries autorisées `nmaxser`. Si l'une de ces conditions n'était pas respectée, l'exécution s'arrêterait et un message d'erreur indiquant le problème serait créé dans le fichier `error_log`.

```
!*********************************************************************!
! subroutine dev2B_fr(ser,order)                                      !
!                                                                     !
!*********************************************************************!
subroutine dev2B_fr(ser,order)
  use MSTABLES
  implicit none
  integer,intent(in):: ser(0:MSordmax),order
  integer::WF(MSordmax)=0,WR=0,sc,arg(MSnvt)=0,exp(MSnvp)=0
  integer:: sinf=0,sin2f=0,der=0,res=0,temp=0
  integer:: k,i,j,mm,jj,inde,indr,indf
  integer:: BINOM,NBTERM
  double precision fact,coef


  ! validation

  if(order.lt.0.or.order.gt.MSordmax) then
     MSmessage='error on dev2B_fr: the order (1) is out&
          & of bound'
```

```
     MSintmess=0
     MSintmess(1)=order
     call ERROR
  endif

  do k=0,order
     if(ser(k).lt.0.or.ser(k).gt.MSnmaxser) then
        MSmessage='error in dev2B_fr: the identifier of ser(see (1))&
             & is (2) and is not valid '
        MSintmess=0
        MSintmess(1)=k
        MSintmess(2)=ser(k)
        call ERROR
     endif
  enddo
```

Dans la seconde partie de la sous-routine, ce sont d'autres vérifications plus spécifiques qui sont présentes. En effet, le but de cette partie est d'une part de vérifier que les variables ' e',' r' et ' f' existent, mais aussi et surtout de déterminer leur index, c'est-à-dire la position qu'elles occupent dans la déclaration des variables. Pour ce faire, le MSNam effectue une boucle sur le nombre de variables (polynomiales pour ' e' et ' r', trigonométriques pour ' f') afin de vérifier l'existence de ce nom de variable. Dès qu'un nom est trouvé, on passe à la recherche suivante, et si une seule des trois variables devait ne pas exister, le MSNam renverait une erreur dans le fichier `error_log`.

```
  ! ************************************************************************
  !
  !  Find the indexes of  "e" the eccentricity,
  !  "r" the ratio r/a,
  !  "f" the true anomaly
  !
  ! ************************************************************************

  do inde=1,MSnvp
     if(MSnamevp(inde).eq.'   e') goto 100
  enddo

  MSmessage='error on dev2B_fr: no polynomial variable is named e'
  MSintmess=0
  call ERROR

100 do indr=1,MSnvp
     if(MSnamevp(indr).eq.'   r') goto 200
  enddo

  MSmessage='error on dev2B_fr: no polynomial variable is named r'
  MSintmess=0
  call ERROR

200 do indf=1,MSnvt
     if(MSnamevt(indf).eq.'   f') goto 300
  enddo

  MSmessage='error on dev2B_fr: no trigonometric variable is named f'
```

```
  MSintmess=0
  call ERROR
```

Après toutes ces vérifications d'usage, la sous-routine proprement dite commence. Puisque l'algorithme de transformée de Lie est récursif comme nous l'avons vu avec la formule (6.11), sa version implémentée commence donc par le cas de base, représentée ici par le cas « `order=0` ». Comme indiqué en commentaire dans le code, il s'agit du cas « où on ne fait rien », c'est-à-dire qu'il suffit de poser l'exposant de ' r' à zéro. Pour cela, il suffit de récupérer les données via la sous-routine `FETCH` puis de modifier l'exposant de ' r' et enfin de le stocker à nouveau.

```
  ! ********************************************************************
  !
  !  If order = 0, return after setting the value of 'r' to 1
  !
  ! ********************************************************************

300 if(order.eq.0) then
     do k=1,NBTERM(ser)
        call FETCH(ser,k,sc,arg,exp,coef)
        exp(indr)=0
        call STORE(temp,sc,arg,exp,coef)
     enddo
     call ERASE(ser)
     call RENAMEE(ser,temp)
     return
  endif
```

La partie qui suit consiste à créer les génératrices qui vont définir le développement en série de Lie. Contrairement à ce que nous avons fait dans les sections précédentes, les génératrices $WR = -\cos f$ et $WF = \dfrac{4\sin f + e\sin 2f}{2(1-e^2)}$ vont être multipliées par e. Cela est dû au fait que, dans les applications numériques, au lieu de considérer e comme le petit paramètre ϵ, ϵ est souvent fixé à 1 et e est considéré comme indépendant et suffisamment petit pour assurer la décroissance des termes avec l'ordre. De plus, lorsque nous faisons les calculs à la main, nous calculons uniquement les fonctions intermédiaires $f_j^{(i)}$ et nous rajoutons les termes en excentricité (e, $\dfrac{e^2}{2}$, $\dfrac{e^3}{3!}$, etc.) par la suite. Point de vue algorithmique, cela n'est pas possible, c'est pourquoi ϵ est fixé à 1 et e est incorporé dans les fonctions. Par ailleurs, puisque le cas `order=0` est traité comme cas de base, il est normal de ne pas avoir de terme indépendant de e dans les génératrices.
La création des fonctions génératrices est ensuite très simple, il suffit de stocker $-e\cos f$ dans `WR` d'une part, et de créer deux fonctions intermédiaires `sinf`$= 2\sin f$ et `sin2f`$= \frac{1}{2}\sin 2f$ d'autre part. Ces deux fonctions sont ensuite utilisées pour créer `WF`, où à chaque étape on vérifie la parité de l'ordre `k`. Dans le cas impair, on ajoute `sinf` et dans le cas pair, on ajoute `sin2f`. Moyennant une multiplication par la factorielle et par e à la puissance souhaitée (afin d'inclure les termes en ordre de e comme dit précédemment), nous retrouvons l'expression (6.27) dans `WF`.

```
  ! ********************************************************************
  !
  !  Formation of the generator of the Lie transform which defines
  !  the expansion
  !
  !  dr/de = -cos f  =  RW/e
  !
  !  df/de = (4sin f +e sin 2f)/2(1-e^2) = FW/e
```

```
!
! ************************************************************************

arg=0
exp=0
arg(indf)=1
exp(inde)=1
call STORE(WR,0,arg,exp,-1.d0)

exp(inde)=0
call STORE(sinf,1,arg,exp,2.d0)
arg(indf)=2
call STORE(sin2f,1,arg,exp,0.5d0)

fact=1.d0
do k=1,order
   if((k-1).gt.0) fact=fact*dble(k-1)
   if(k.ne.2*(k/2)) then
      call ACUM(sinf,WF(k),fact)
   else
      call ACUM(sin2f,WF(k),fact)
   endif

   call XMON(WF(k),'   e',k)
enddo

call ERASE(sinf)
call ERASE(sin2f)
```

La partie qui suit est le coeur de la sous-routine proprement dit. En effet, elle consiste en l'application de l'algorithme, comme nous l'avons fait à la main, et comme indiqué dans [Henrard, 2000]. Pour chaque contribution des dérivées par rapport à ' r' et ' f', le programme ajoute d'abord le terme de la série au résultat via `ACUM` (terme qui a été au préalable multiplié avec `SCALE`) puis calcule la dérivée du terme précédent et le stocke dans `der` via `PDERP` pour ' r' ou `PDERT` pour ' f'. On ajoute enfin au résultat le produit de `der` et de la génératrice pour effectuer la dérivée de Lie. Le reste de la sous-routine n'est qu'un simple jeu d'indices.

```
! ************************************************************************
!
!  Expansion of "ser" by Lie transform
!
! ************************************************************************

! scaling the input series

fact=1.d0
do k=1,order
   fact=fact*dble(k)
   call SCALE(ser(k),fact)
enddo

! contribution of the derivatives w.r.t "r"

do k=1,order
```

```
    do i=k,order
       mm=order+k-i
       call ACUM(ser(mm),res,1.d0)
       call PDERP(ser(mm-1),der,'   r')
       call PROD(der,WR,res,1.d0)
       call ERASE(der)

       ! contribution of the derivatives w.r.t. "f"

       jj=mm-k+1
       do j=1,jj
          fact=dble(BINOM(mm-k,j-1))
          call PDERT(ser(mm-j),der,'   f')
          call PROD(der,WF(j),res,fact)
          call ERASE(der)
       enddo
       call ERASE(ser(mm))
       call RENAMEE(res,ser(mm))
    enddo
    call ERASE(WF(order-k+1))
  enddo
  call ERASE(WR)
```

La dernière partie de la sous-routine est la renormalisation. Elle consiste à poser l'exposant de ' r' à zéro ainsi qu'à diviser le résultat final par la factorielle. Nous remarquons également qu'il s'agit du moment choisi pour négliger les termes dont le coefficient est plus petit que `MSaccurary` via l'utilisation de `CUTEPS`.

```
  ! ************************************************************************
  !
  !  Renormalisation: set to zero the exponent of r
  !   divide by the factorial
  !
  ! ************************************************************************

  fact=1.d0
  do k=0,order
     do i=1,NBTERM(ser(k))
        call FETCH(ser(k),i,sc,arg,exp,coef)
        exp(indr)=0
        call STORE(res,sc,arg,exp,coef*fact)
     enddo
     call ERASE(ser(k))
     call RENAMEE(ser(k),res)
     call CUTEPS(ser(k),MSaccuracy)
     fact=fact/dble(k+1)
  enddo

end subroutine dev2B_fr
```

Nous avons ainsi pu observer la structure, le fonctionnement mais aussi l'intérêt d'une sous-routine de la sorte. Un futur utilisateur pourrait développer de nouvelles sous-routines de la sorte, comme cela a été fait avec `dev_angle` qui n'était pas présente dans la version originale du MSNam. Cela démontre bien l'énorme potentiel existant dans le MSNam.

Chapitre 7

Comparaison avec un autre manipulateur : Chronos

Au cours de ce travail, nous nous sommes rendus à Milan auprès du professeur Antonio Giorgilli et nous avons eu l'occasion d'utiliser un autre manipulateur symbolique « maison », nommé Chronos, créé par ce dernier.

Durant trois mois, nous avons pu découvrir Chronos et apprendre à l'utiliser. Dans ce chapitre, nous n'allons pas faire l'éloge de l'un ou l'autre manipulateur, mais simplement comparer certains de leurs aspects.

Afin de partir sur de bonnes bases, nous allons commencer par donner un descriptif de Chronos, sous forme de manuel pour l'utilisateur. En effet, contrairement au MSNam, Chronos possède déjà (partiellement en tout cas) un document servant de descriptif. Cependant, selon les dires du professeur Giorgilli ce descriptif ne permet pas « d'apprendre à apprivoiser Chronos ». C'est pourquoi il nous a chargé d'écrire un manuel qui, couplé au descriptif existant, permettra aux nouveaux utilisateurs une prise en main plus aisée.

Ce manuel, écrit en anglais afin d'assurer bonne compréhension de tous, fait l'objet de la section suivante.

7.1 Chronos Manual

This document's aim is to give an introduction about how to use Chronos if you're a new user. Remember that Chronos is a library of routines written in C the purpose of which is symbolic and algebraic manipulations. However, it should not be intended to be a general purpose manipulator such as, e.g., Mathematica or Maple. It is rather specialized for perturbation expansions such as those that occur in Celestial Mechanics and in KAM or Nekhoroshev's theory.

We're not going to give a full explanation about how Chronos works for two obvious reasons. The first because this is supposed to be done in the full Chronos description (the one we'll talk about later) and the second because it's nearly impossible to explore all possibilities in such a paper. Actually we're going to give the new user keys to begin programming with Chronos.

In the first section, we're going to explain how to install Chronos and where you can find the files you need.

Afterwards, we're going to explain how to use some basic subroutines of Chronos such as initializing Chronos, storing a function, making products or other basic operations, release memory, etc.

Finally, in the last part, we're going to include some advice about some tricky things or common errors you have to avoid.

7.1.1 Before using Chronos

At the moment, the latest released version of Chronos is `chronos_0.65`. If you want to use Chronos, which is split into different folders, you first need to install it. As the version number clearly indicates, it is still far from a version 1.0 ready for diffusion.

In order to do it, we refer to the file `installation.txt` given in appendix. Basically, untar the distribution kit and the the directory where chronos has been installed, using, e.g.,

```
export CHRONOS=/home/myname/documents/chronos_0.65
```

or add it in your `.bashrc` file. This is important for compiling. If you want to verify if the location is right, use

```
env | grep CHRONOS.
```

Afterwards, you need to create some data file, but we refer to appendix which contains the detailed installation procedure.

Now you're ready to use Chronos, but first of all, we'll give you more information about the contents of every folder :

- `doc` contains the documentation of Chronos, especially the file `Chronos.dvi` which is the full description (but not yet completed) of Chronos. We'll refer many times to this file for more information.

- `src` contains the source code for all functions and subroutines. The file `chronos_def.h` is the file containing all templates. If you look for a function you know that exist, use this file. It is split into parts defined by comments of the type `/* part */` that correspond to a `.c` file containing the functions related to `/* part */`. For example, if you look for `scalar_multiply`, you can see that it is located in the `/* algebr */` part. That means that if you want more details about the function `scalar_multiply`, you have to look into the file `algebr.c` which, in fact, contains all functions related to algebraic manipulations.
  This folder also contains the files `chronos_cmn.h`, `chronos_ext.h` and `chronos_msg.h` which respectively define the types, structures and error messages used by Chronos.

- `local` contains definitions concerning the local directory structure for Chronos type definitions for variables. The purpose is to define the internal names for `typedef` definitions controlling the length of data (mainly integers). The choice mainly depends on the computer's architecture and/or compiler's characteristics. The current version is adapted for use on Intel or AMD CPU's with 64 bit architecture, and works with the GNU C-compiler. Adapting the local definitions to different processors and/or compilers may require some skill.

- `util` contains file used to create the Chronos data file. See appendix for more details.

- `msc` stand for miscellaneous. It is the folder in which you can find the file `installation.txt` but also the logbook, in which you can find the evolution between Chronos versions.

- `lib` is a folder created after installation, and contains the compiled library.

7.1.2 How to use Chronos

Initializing and closing Chronos

In order to use Chronos, create a new file. Let us name it `test.c`. First of all, you need to include some header files, namely `chrlocal.h, chronos_msg.h,chronos_cmn.h, chronos_def.h` and `chronos_ext.h`. Moreover, if you want to use standard packages, we recommend also to include `math.h,stdio.h,stdlib.h` and `time.h`. Also define the number of variables you need (say `NDIM = 4`), in order not to forget or change it.

As an example, let us consider a polynomial. It is characterized by the number of variables `NDIM` and its minimal degree `smin` and maximal degree `smax`. The exponents of each variable are stored in a vector `k`. The coefficients `coef` can either be integer, double, complex or interval, we refer to chapter 6 of Chronos description for more details about coefficient's types. Here we consider double real coefficients. The user can easily switch to other types.

Begin your main program and define every variable you need. In order to create and manipulate a function, you need to create these variables :

- `int i` : A counting variable.
- `DOUBLE coef` : A double precision coefficient.
- `INT2 k[NDIM]` : A vector containing the exponents of the variables of a term.
- `UNS2 perm[NDIM]` : A vector which gives an order to the variables, used for example to know which variable is conjugated to another.
- `UNS2 type[NDIM]` : A vector giving the types of the variables. As we can see further, this vector will successively contains the types of the variables (such as polynomial or fourier) but also the types of the genereting function and canonical structure.
- `UNS2 count[NDIM]` : A vector giving the number variables of each type. In fact, a function can contain either polynomial and fourier types, as in Poisson series for example. This vector basically says the size of each block of variables (see next section for explanations).
- `INT2 smin[NDIM]` : A vector giving the minimum order of the variables.
- `INT2 smax[NDIM]` : A vector giving the maximum order of the variables.
- `DMHDR *idxfr, *derfr, *prodfr, *genfr, *canfr` : Variables needed to define the structure of a function.
- `DMHDR *fdmh` : A pointer to function (at least the adress for the moment). Create as many as needed.
- `FN_HANDLERS fh` : A structure function handlers, which basically will remember the structure of a function.

After that, the main program begins. Use `chr_init()` to initialize Chronos and `chr_finish()` to close it.

So the structure of a program using Chronos should be

```
#include <stdio.h>
#include <math.h>
#include <time.h>
#include "chrlocal.h"
#include "chronos_msg.h"
#include "chronos_cmn.h"
#include "chronos_def.h"
#include "chronos_ext.h"

#define NDIM 4 /* total number of variables */

int main()
{
  int i;
```

```
  DOUBLE coef;

  INT2 k[NDIM];
  UNS2 perm[NDIM], count[NDIM], type[NDIM];
  INT2 smin[NDIM], smax[NDIM];

  DMHDR *idxfr, *derfr, *prodfr, *genfr, *canfr;

  DMHDR *fdmh;

  FN_HANDLERS fh;

  /* initialize Chronos */
  chr_init();

  /* Main part of the program*/


  /* close Chronos */
  chr_finish();

}
```

Storing a function

Now it's time to learn about how to create a function. In fact, there are two possibilities, inserting it by yourself or reading it from a file. We'll show you the first method, and give you some keys and references for the second.

We assume that you already have created the variables defined earlier and we begin with the way to create the structure of a function.

Let us suppose that we only have one component in the function, which in our case, means that we only have polynomial variables. So we can fix `count[0]=NDIM` which mean that we only have one block. It's indexing type is polynomial, so we fix `type[0]=IDX_POLYNOM`.
If, for example, we had two polynomial and two trigonometric variables, we should have done something like :

```
type[0]=IDX_FOURIER
type[1]=IDX_POLYNOM
count[0]=2
count[1]=2
```

Which would mean that we have two blocks of two variables, the first block is trigonometric and the second is polynomial.

In our case, we also fix the permutation fragment as `perm=[1 2 3 4]` but in some cases, such as action-angles variables, it's important to respect a fixed order.
Moreover, since we have only one block, let us say that our polynomial is of minimum order 0 and maximum order 5, writing `smin[0]=0` and `smax[0]=5`.

Now we can define the indexing fragment
`idxfr = fn_def_idx_fragm(NDIM, perm, 1, count, type, smin, smax, NULL, NULL)`.
It basically says to Chronos "How to store".
Afterwards, we have to define the derivative, product, generating and can fragments in the same way. They

respectively say to Chronos, in our case, that derivatives and products are of the type polynomial and that we give no rule for computing generating functions or canonical structure. So we must have something like :

```
/* permutation vector: order of the variables (for the moment just
   consider the order 1,2,...) */
for(i = 0; i < NDIM; ++i)
  perm[i] = i+1;
/* 1 fragment of dimension NDIM */
count[0] = NDIM;
/* of polynomial type */
type[0] = IDX_POLYNOM;
/* with degree at least 0 */
smin[0] = 0;
/* and at most 5 */
smax[0] = 5;
/* ok, in this way we say to Chronos "how to store" */
idxfr = fn_def_idx_fragm(NDIM, perm, 1, count, type, smin, smax, NULL, NULL);
/* well, it's not yet finished... */

/* the derivatives are polynomial */
type[0] = DER_POLYNOM;
derfr = fn_def_fragm(NDIM, perm, 1, count, type);
/* the products are polynomial  */
type[0] = PROD_POLYNOM;
prodfr = fn_def_fragm(NDIM, perm, 1, count, type);
/* no rule on "how to compute the generating functions" */
type[0] = GEN_UNDEF;
genfr = fn_def_fragm(NDIM, perm, 1, count, type);
/* no canonical structure */
type[0] = CAN_UNDEF;
canfr = fn_def_fragm(NDIM, perm, 1, count, type);

/*  Create the wanted polynomial  */
fdmh = fn_create(NDIM, 0, idxfr, derfr, prodfr, genfr, canfr, FN_BIN_TREE, CF_DOUBLE);

/* free the memory, we do not need anymore these stuff */
dm_release(idxfr);
dm_release(derfr);
dm_release(prodfr);
dm_release(genfr);
dm_release(canfr);
```

Where we can see that we freed the memory after creating the polynomial with `fn_create`. For more details about types and structures, we refer to the descriptions in the documentation `Chronos.dvi`.

Until here, we only created the structure of the function. The way to store the coefficients and exponents of our polynomial is much more simple. Let's suppose that we want to store the polynomial

$$\frac{1}{2}(x_1^2 + y_1^2) + \frac{1}{2}(x_2^2 + y_2^2)$$

You first need to get the function handlers by doing `get_fn_handlers(fdmh,&fh)` and then create a string that will contain the coefficient using `char xcf[fh.coefsize]`. It's length depends on the handler.

Then just introduce your coefficients and exponents and transform the coefficient into a string using `fh.cv_from[BCFTYP_DOUBLE](&coef,xcf` and the add the term into your function, using `fh.add_coef`(xcf,k,fdmh). You must have something like that :

```
/*  get the handlers of the function */
get_fn_handlers(f1dmh, &fh);
{
  /* create a "string" that will contain the coefficient.*/
  char xcf[fh.coefsize];
  /* set to zero the vector of the exponents (remember in C the
     arrays are not set to zero by default) */
  for(i = 0; i < NDIM; k[i++] = 0);
  for(i = 0; i < NDIM; ++i) {
    /* set the i-th variable exponent to 1 */
    k[i] = 2;
    /* set the coefficient to 1.0 */
    coef = 0.5;
    /* convert from double to internal Chronos representation  */
    fh.cv_from[BCFTYP_DOUBLE](&coef, xcf);
    /* add the coefficient in the function. */
    fh.add_coef(xcf, k, fdmh);
    /* reset the i-th variable exponent to zero */
    k[i] = 0;
  }
}
```

Note : we should have used `fh.put_coef` instead of `fh.add_coef`. The difference is that `put_coef` overwrites an existing coefficient (if any), while `add_coef` adds the new coefficient to the existing one. If the coefficient is still undefined, then both operations produce the same result.

It may seem very difficult at the beginning, moreover because this example is simple. But using it as an example is nearly enough to create every type of function.

The other way to store a function is to use a file and import data from it. For that, you need a file `myfct.asc` containing the coefficients and another file `myfct.desc` containing the description of the structure. Then, just import the function using `fdmh=fn_import_ascii("myfct.asc")`.
To see how it works, we refer for example at the folder <xxx>/etest/alg which contains such a file.

Compiling

In order to compile the file you created, it is convenient to create a `makefile`. This file has a particular structure, giving paths at the beginning and then compiling using "standard" `gcc` compiler. A `makefile` has the structure below

As you can see, it firstly includes the paths to Chronos library and then you can see the `gcc` commands. To use it, simply type

`make all`

in a terminal to compile all file, or

`make test`

to compile juste the file `test.c` for example. You can also use the command `make clean` to delete all executable and temporary file.

Note : `CFLAGS` defines the compiler's options. The debugging is included, but may be removed or replaced by code optimization whe the program is ready for use.

Some basic manipulation

Once you've stored a function, you obviously want to play with it. I mean, you want to make some algebraic manipulations for example. In this section, we'll show you some examples of basic manipulations that you can do with a function. It would be very useful if you want to make some more complex manipulations afterwards.

Let's begin with functions that are located in the file `algebr.c`, which basically contains all functions dealing with algebraic manipulations, such as sum, products, derivatives, ...

Remember that our function is stored as a `DMHDR`, which is the structure of a function. You can remark that there are functions that return a `DMHDR` values and others that return no value (`void`) or integer value.

In the first case, as for example in
`extern DMHDR *sum(DMHDR *fdmh,DMHDR *gdmh)`,
it means that it creates a new function. In this case, given the functions `fdmh` and `gdmh`, the routine `sum` creates a new function which is the sum of the other two. You have to call this routine in this way :

```
DMHDR *zdmh;
zdmh=sum(fdmh,gdmh)
```

Since `zdmh` is declared, the routine will create all the structure and store all new coefficients.

In the second case, as for example in
`extern int add_to(DMHDR *srcdmh,DMHDR *dstdmh)`,
the routine returns an integer, which is a flag that says "if the routine fails or not". In this case, give two existing functions `srcdmh` and `dstdmh`, the routine `add_to` adds to the destination function `dstdmh`, the terms of the source function `srcdmh`. For the user, what is interesting is the fact that the destination function in modificated afterwards. The integer flag is not so important, it "just" gives some information about how to process has gone.
Depending on the importance you give to this flag, you can call the function in two ways :

```
add_to(srcdmh,dstdmh)
```

or

```
l=add_to(srcdmh,dstdmh).
```

You can also remark that some routines requires other types, or no type at all, so always be careful of that before using one.
We also add a comment about what we call "do"-routines. These routines are the ones that actually do the operation. They are automatically called in the other routines. The user have to use classical routines that cares on warning and handling errors and never use "do"-ones unless he really knows what he is doing.

Releasing memory

As we said before, Chronos uses dynamic memory handling. It means that, as you must do everytime you allocate memory, you have to release memory by deleting things you don't need anymore.

In previous part, we saw for example that we released memory after creating a function by using `dm_release` to delete the fragment's definitions we didn't need anymore.

In the same way, anyway you don't need anymore a function, especially if you use temporary ones, you have to use `fn_release`. Usually, you delete temporary functions as soon as you don't need them anymore,

and delete the rest at the end of the program.

By the way, the easiest way to see if you effectively released all memory is to ask Chronos to give you some information about the working process. Just add these two lines at the end of your programme, before closing Chronos :

```
dm_shuffle(0);
db_list_summary();
```

As information, shuffling is performed by `dm_shuffle`. It's purpose is to compact all used movable memory blocks at top of the dynamic memory and to merge all contiguous free memory block into one.

The second line is the one that gives some information about memory use such as how many headers you used, etc.

```
Headers: free_dmh 14998, free 1, used 0, fixed 1; dm: free 67093864, used 0, fixed 15000.
Maximum of used memory blocks: 16316 out of 67108864 (0.02%)
```

It is very important to have "0" in used, which means that you released everything well. As we will see in next section, some mistakes make it impossible.

Note : Chronos will clean memory anyway when you call `chr_finish`. However, it is a good practise to do it explicitly, for it may help in discovering possibly huge data that are lost causing a waste of memry and sometimes failures at exectution time due precisely to lack of memory.

7.1.3 Some common mistakes to avoid

In this section we will give you some advice about common mistakes done by unexperimented users. As first example we will show you the problem of increasing degrees of a polynomial.

When you use a routine that returns a new function, such as `prod`, all the structure of this new function is created automatically. It means that if your two functions where of maximum degree 5, the result is of maximum degree 10. It's quite logical and you may not see the problem. In fact, unexperimented users may define a polynomial of maximum degree n just "to be sure it's enough". There is the mistake, you must create polynomials of maximum degree as low as possible (if, of course, you know it). It avoids creating functions of exponentially increasing degrees. Actually, use the minimum degree as high as possible for the same reason.

Moreover, if you're in the particular case of homogeneous polynomials, use type `HOMOPOL` instead of `POLYNOM` and define `smax=smin`. The less space you use for storing function, the less memory you waste!

On the one hand, we also recommend to use routines that returns a `DMHDR` instead of `void` as usually as possible. For example it's better doing

```
zdmh=prod(xdmh,ydmh)
```

than creating a empty function `zdmh` and adding the product by doing

```
add_prod_to(xdmh,ydmh,zdmh).
```

Moreover, regarding to the previous remark, if you have created your empty function `zdmh` by a duplicating routine such as `fn_duplicate`, you may have created a structure that does not correspond to the real one. Imagine you have done

```
zdmh=fn_duplicate(xdmh)
add_prod_to(xdmh,ydmh,zdmh),
```

you obviously create `zdmh` as a polynomial which has the same degree as `xdmh` and then add in it a polynomial of higher degree! In fact, the result is correct, but the method isn't. Try to avoid that.

In the other hand, the opposite mistake also exist. Let's imagine that you have two functions `xdmh` and `ydmh` and that you want to multiply them. In many algorithms, you can see something like

$x \leftarrow x * y$

which means that you have to replace x by the product $x * y$. A statement such as

```
xdmh=prod(xdmh,ydmh)
```

has a dangerous side effect : the pointer `xdmh` is redefined so that it points to the result, but the previous pointer is "lost". That means that the old function still exists and uses memory ressources, but in no way you can release it. You may suspect that this happend if after releasing everything and calling `db_list_summary` you discover that there are still used memory blocks. In this cas you must use

```
add_prod_to(xdmh,ydmh,xdmh)
```

if your are sure that the result can be stored (I mean if `xdmh` and `ydmh` are polynomials of degree 3, the result is of degree 6 and you must be sure that `smax` is at least 6 for `xdmh`. Otherway, use a temporary function.

In the first case, `xdmh=prod(xdmh,ydmh)` creates new handlers for function `xdmh` so that you can no more free memory for the old ones. It means that you will always remain with used handlers at the end, even if they aren't used, because they became unaccessible.

7.1.4 Permutations, generating function and canonical structure

In the section about creating functions, you may have noticed that we have defined the permutation vector as identity $(1, 2, 3, 4)$ and that we have set `GEN_UNDEF` and `CAN_UNDEF` respectively in the `GEN` and `CAN` fragments.

Basically, the permutation vector gives an order to the variables. It is important for two reasons. The first one is about wasting memory. In fact, function's coefficients are stored in binary trees. The adress of a function points to the first element of the tree, which is 0 or 1. Regarding to its value, each element points to the second element of the tree, which is 0 or 1, etc. Chronos is made in the way that if, for some reason, the first element is 0, then no memory is allocated for this part of the tree.

Let us suppose that you have set the permutation vector as identity as we did before, but because of some reason, the exponent of the last variable is always even. It means that all the binary tree is created, but the last element is always 0. If instead you define the permutation vector such as that variable becomes the first, half the tree doesn't need to be allocated ! This is a little bit tricky and seems obsolete regarding to the large amount of memory available currently.

The second reason, and more obvious reason for common use is that the permutation vectore defines which variable is conjugated to another. Usually, we define first variables as coordinates and next ones as moments. If you want to use some routines that requires to know the canonical structure, such as Poisson bracket, you need to define a `CAN` fragment. Both this one and the permutation vector give Chronos the information about canonical structure. For example, $(1, 2, 3, 4)$ means that variable 1 and 2 are respectively conjugated with 3 and 4.
A classical example occurs when you use action-angle variables. In fact, angles are related to coordinates and actions to moments. It means that angles must be "before" actions in the permutation vector. But usually, we write this variables in the other way, with first actions and then angles. So you can define it in the usual way and set permutation vector to $(3, 4, 1, 2)$ in order to inform Chronos to use the reverse order.

The `GEN` fragmentation table is related to the opration of giving a system a so called "normal form". The `GEN` table may take different forms depending on the problem.

For more information and details about all these structures, we refer to chapter 6 of Chronos description, which is quite complete on the subject.

7.1.5 Appendix : Installation guide

The information here refers to the installation procedure for version 0.65 of Chronos. Updated information for more recent releases is expected to be found in the file `installation.txt` located in the `msc` folder.

```
Hints for installing Chronos on a Linux system


1. Untar the distribution kit:

tar -xvzf chronos-x.yy.tar.gz

   (replace x.yy with the current version number) this will create a
   directory ./chronos with the complete structure of
   subdirectories. E.g., assuming that you are installing Chronos
   under the directory /home/drillo you will find a directory
   /home/drillo/chronos containing all files of the package.

2. Define the global symbol CHRONOS so that it points to the directory
   where chronos has been installed. E.g., type the commands

CHRONOS="/home/drillo/chronos"
export CHRONOS

In order to make the definitions permanent put these commands in your
.bashrc file.


3. Check that there is no file
 <xxx>/chronos/lib/libchronos.a.
 <xxx>/chronos/util/chrdat.o.
   If any of them exists, delete it.


4. Create the Chronos data file with the command:

cd <xxx>/chronos/util
make

   An output similar to the following is expected:

gcc -c -g -I. -I/home/antonio/chronos/src -I/home/antonio/chronos/local chrdat.c
gcc -g -o chrdat -I. -I/home/antonio/chronos/src -I/home/antonio/chronos/local chrdat.o -lm
./chrdat
Chronos V 0.53:16
      File chronos.dat, created on  7- 4-1999 10:32:58
 Number of error messages : 98
 Total number of messages : 137
rm /home/antonio/chronos/src/*.o
rm: /home/antonio/chronos/src/*.o: No such file or directory
make: *** [chrdat] Error 1

   Do not worry about the last error line: it just means that an
```

```
   attempt to delete nonexisting files has been made; so it's OK.

5. Create the library:

cd <xxx>/chronos/src
make

   An output similar to the following is expected:

gcc -I/home/antonio/chronos/src -I/home/antonio/chronos/local -c -g  chronos_cmn.c
gcc -I/home/antonio/chronos/src -I/home/antonio/chronos/local -c -g  general.c
gcc -I/home/antonio/chronos/src -I/home/antonio/chronos/local -c -g  messer.c
gcc -I/home/antonio/chronos/src -I/home/antonio/chronos/local -c -g  dmhandler.c
gcc -I/home/antonio/chronos/src -I/home/antonio/chronos/local -c -g  asciibin.c
....... (more similar lines)......
ar -rsv /home/antonio/chronos/lib/libchronos.a chronos_cmn.o general.o
messer.o dmhandler.o asciibin.o dmdebug.o arithmetic.o idxhandler.o
idxpol.o idxtrg.o idxhompol.o slhandler.o slhdebug.o idxdebug.o
fndebug.o fn_oldver.o fn_usage.o fn_gen.o fn_frtab.o fn_frag.o
fn_bintree.o  fn_list.o fn_util.o cf_double.o algebr.o algdebug.o
alg_pol.o alg_trg.o genfun.o imatrix.o idxcyl.o idxdalemb.o
a - chronos_cmn.o
a - general.o
a - messer.o
a - dmhandler.o
a - asciibin.o
a - dmdebug.o
....... (more similar lines)......



Installation is complete. You may perform a few tests in order to
check that everything works properly.


6. Executing a test program.
   There  are a lot of <yyy>.c files in the directory
   <xxx>/chronos/test .  You may execute any of them by typing the
   command (first cd <xxx>/chronos/test)

./try <yyy>

   Do *not* add .c to the filename.
   Deciding if the test has produced correct results is up to
   you. Usually the test programs terminate with a message

Test completed.

See the all file in directory msc for examples
```

7.2 Complexité des deux manipulateurs

Après cette introduction à l'utilisation de Chronos, nous allons nous intéresser aux principales différences entre les deux manipulateurs. Outre le fait qu'ils sont écrits dans deux langages différents, ce qui n'aide pas pour faire une comparaison, nous allons dégager une première grande différence entre le MSNam et Chronos : leur complexité.

Par complexité, nous entendons la quantité de choses que chaque manipulateur est capable de faire. Et là, il faut se rendre à l'évidence, Chronos est bien plus complexe que le MSNam.

D'une part, alors que le MSNam se restreint au traitement des séries de Poisson, Chronos permet de manipuler un bien plus grand nombre d'objets tels que les polynômes, les séries de Fourier, les fonctions cylindriques (fonctions de Bessel) ou les polynômes trigonométriques de D'Alembert. De plus, les coefficients de ses séries peuvent prendre d'autres types que les réels en double précision. Chronos permet notamment d'utiliser les variables complexes ou les intervalles.

D'autre part, la librairie de Chronos est bien plus fournie que celle du MSNam. Là où le MSNam compte exactement soixante sous-routines ou modules, Chronos en dénombre plus de cinq cents.

Ces deux données mettent certainement en avant le fait que Chronos est plus complet que le MSNam. Mais, il ne faut pas oublier que cela vient de la volonté même de leur créateur respectif. Jacques Henrard voulait un manipulateur pour la mécanique céleste, alors qu'Antonio Giorgilli voulait quelque chose de beaucoup plus général, qui peut servir en mécanique céleste. C'est pourquoi nous insistons une fois de plus qu'il n'est nullement question de faire le procès de l'un ou l'autre manipulateur.

De plus, il faut garder à l'esprit que, dans la majeure partie des cas, plus un programme est complexe, plus il est compliqué à prendre en main. Nous pouvons nous en rendre compte dans un simple fait : le nombre d'utilisateurs de Chronos est aussi faible que celui du MSNam. C'est principalement pour cela que la partie précédente a été rédigée. Nous l'avons d'ailleurs basée sur nos propres difficultés.

7.3 Gestion dynamique de la mémoire

Dans la myriade de différences entre les deux manipulateurs, nous allons dégager une caractéristique très particulière existante dans Chronos et pas dans le MSNam, il s'agit du stockage dynamique.

En effet, nous avons vu au Chapitre 2 que le MSNam utilise des tableaux statiques dont la taille est définie dans le module `parameters`. En revanche, les explications du manuel susmentionné parlent bien d'une allocation dynamique de mémoire.

Bien que les deux méthodes de stockage des données soient basées sur les arbres binaires, c'est l'allocation dynamique qui fait la différence. Les tableaux utilisés par le MSNam sont généralement remplis d'une grande quantité de 0, allouant ainsi de la mémoire pour rien. De son côté, Chronos ne crée les branches de l'arbre binaire qu'au moment où elles sont nécessaires. Ainsi, comme nous en parlions à la section 7.1.4, si pour une raison quelconque nous savons que la valeur d'une variable est toujours paire, la placer en première position permet de ne définir que la moitié de l'arbre binaire. En effet, cette variable ayant toujours un « 0 » dans l'arbre, il n'est pas nécessaire d'allouer de la mémoire pour la partie correspondant au « 1 ». Ce dernier cas n'est pas forcément fréquent, mais permet d'épargner une grande quantité de mémoire le cas échéant.

Il serait intéressant que de futurs travaux déterminent s'il est possible d'utiliser une méthode semblable pour le MSNam en utilisant les tableaux dynamiques « `allocatable` ».

7.4 Application

Afin d'illustrer quelque peu nos propos, nous avons décidé d'effectuer à nouveau l'application du Chapitre 5 avec Chronos. Puisque la librairie de Chronos n'inclut pas de fonction permettant d'évaluer une série ni d'intégrateur numérique, nous illustrerons uniquement la lecture d'une série, la multiplication par un scalaire et l'usage des dérivées.

Le code ci-dessous effectue plusieurs tâches succesives, tout comme nous l'avons fait avec le MSNam au Chapitre 5. Tout d'abord, après la déclaration des variables utiles à Chronos et des autres paramètres nécessaires pour l'application, nous utilisons la fonction `fn_import_ascii` pour récupérer les données contenues dans le fichier. Nous reviendrons sur cette partie par la suite.
Ensuite, nous utilisons la fonction `scalar_multiply` pour multiplier les coefficients par la constante `cst`. Nous avons ensuite décidé d'afficher la série à l'écran via l'utilisation de `fn_print_coef`.
Par la suite, nous dérivons la série par rapport à chaque variable grâce à la fonction `deriv`, puis nous effectuons la division par $\pm\frac{1}{L_i}$ à nouveau grâce à la fonction `scalar_multiply`.
Enfin, nous effectuons la division par E_i. Cependant, il n'existe pas de fonction semblable à `XMON` dans la librairie Chronos, nous devons donc effectuer la modification à la main. Pour se faire, nous utilisons les fonctions `fh.first_nz` et `fh.next_nz` pour parcourir la série et modifier les exposants.
Nous affichons ensuite les résultats des dérivées, puis nettoyons la mémoire de manière classique.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "chrlocal.h"
#include "chronos_msg.h"
#include "chronos_cmn.h"
#include "chronos_def.h"
#include "chronos_ext.h"

#define NDIM 4 /* total number of variables */

int main()
{
  int i,l;
  DOUBLE coef,cst,L1,L2;
  INT2 k[NDIM];

  DMHDR *fdmh,*H[NDIM],*der[NDIM];
  const double pi=3.14159265358979323;
  double m0,m1,m2,a1,a2,G,e1,e2,w1,w2;

  FN_HANDLERS fh;

  /* initialize Chronos */
  chr_init();

  m0=1.31;
  m1=1.92/1047.355;
  m2=4.13/1047.355;
  a1=0.832;
  a2=2.53;
  G=4*pow(pi,2);
```

```
cst=-G*m1*m2/a2;
L1=m1*pow(G*m0*a1,0.5);
L2=m2*pow(G*m0*a2,0.5);

//CI
e1=0.224;
e2=0.267;
w1=250.8;
w2=269.7;

/*Import data */
fdmh=fn_import_ascii("myser.asc");


get_fn_handlers(fdmh,&fh);
CHR_ICF_TY cof[fh.coefsize];

//coef=cst;
fh.cv_from[BCFTYP_DOUBLE](&cst,cof);
scalar_multiply(fdmh,cof);

printf("***** Hamiltonien \n");
fn_printnz_coef(fdmh);

/*Make derivative */
H[0]=deriv(fdmh,0);
H[1]=deriv(fdmh,1);
H[2]=deriv(fdmh,2);
H[3]=deriv(fdmh,3);

fn_remove(fdmh);

/*Muliply by "1/L"  */
coef=-1.0/L1;
fh.cv_from[BCFTYP_DOUBLE](&coef,cof);
scalar_multiply(H[0],cof);
coef=-1.0/L2;
fh.cv_from[BCFTYP_DOUBLE](&coef,cof);
scalar_multiply(H[1],cof);
coef=1.0/L1;
fh.cv_from[BCFTYP_DOUBLE](&coef,cof);
scalar_multiply(H[2],cof);
coef=1.0/L2;
fh.cv_from[BCFTYP_DOUBLE](&coef,cof);
scalar_multiply(H[3],cof);

/* divide by E */
for(i=0;i<NDIM;i++){
  fn_open_read(H[i]);
  der[i]=fn_duplicate(H[i]);
  get_fn_handlers(H[i],&fh);
  {
```

```
      char xcf[fh.coefsize];
      l=fh.first_nz(&coef,k,H[i]);
      while(l>0){
if(i%2==0){
  k[2]=k[2]-1;
}
else{
  k[3]=k[3]-1;
}
fh.cv_from[BCFTYP_DOUBLE](&coef,xcf);
fh.put_coef(xcf,k,der[i]);

l=fh.next_nz(&coef,k,H[i]);
      }
    }
  }


  /*Print results */
  printf("***** der(1) \n");
  fn_printnz_coef(der[0]);

  printf("***** der(2) \n");
  fn_printnz_coef(der[1]);

  printf("***** der(3) \n");
  fn_printnz_coef(der[2]);

  printf("***** der(4) \n");
  fn_printnz_coef(der[3]);

  /*Free memory */
  for(i=0;i<NDIM;i++){
    fn_remove(H[i]);
    fn_remove(der[i]);
  }

  /* some info about memory usage */
  dm_shuffle(0);
  db_list_summary();

  /* close Chronos */
  chr_finish();

  printf("Test completed.\n");

  return(0);
}
```

Pour en revenir à la complexité de Chronos, nous remarquons qu'avant l'utilisation de routines telles que `scalar_multiply` ou `add_coef`, il faut passer en représentation interne de Chronos. En effet, alors que les sous-routines du MSNam sont assez « directes », l'utilisation de types internes avec Chronos nécessite souvent l'utilisation de la fonction `cv_from` pour passer de l'un à l'autre.

Nous avons également noté l'absence d'une fonction de type `XMON` qui, rappelons-le, permet de multiplier une série par un monôme à une puissance donnée.
De plus, alors que le MSNam accepte sans aucun problème les exposants négatifs, il n'est pas possible de les utiliser avec Chronos. Cette caractéristique peut s'avérer assez handicapante dans certains cas.

Revenons maintenant à l'importation des données. Nous avons en effet utilisé `fn_import_ascii` pour récupérer les données dans le fichier. Ce faisant, nous avons contourné la fastidieuse déclaration des types comme expliqué dans le manuel de Chronos. Cette technique nécessite cependant de modifier d'une part le fichier `.dat` en fichier `.asc` et, d'autre part, de créer un fichier formaté contenant les types, sous forme de `.descr`, appelé description de la série.

Dans le cas présent, le fichier se présente sous la forme suivante :

```
# Data and coefficient's type:
2 0
# Number of total and active variables:
4 4
# idx permutation
 1 2 3 4
# Number of idx fragments:
2
# fragment descriptors: length, type, smin, smax
 2 1 0 12
 2 0 0 12
# mu vector :
 0 0 0 0
# nu vector :
 0 0 0 0
# der permutation
 1 2 3 4
# Number of der fragments:
2
# fragment descriptors: length, type
 2 2
 2 1
# prod permutation
 1 2 3 4
# Number of prod fragments:
2
# fragment descriptors: length, type
 2 1
 2 0
# gen permutation
 1 2 3 4
# Number of gen fragments:
2
# fragment descriptors: length, type
 2 0
 2 0
# can permutation
 1 2 3 4
```

```
# Number of can fragments:
2
# fragment descriptors: length, type
 2 1
 2 1
```

Avec un peu d'habitude, il n'est pas trop compliqué de comprendre ce que contient ce fichier, mais lors d'une première utilisation, il vaut mieux lire les données de manière classique puis de les exporter afin de créer automatiquement les fichiers `.descr` et `.asc`. Le code ci-dessous permet d'illustrer cette méthode. Nous y définissons les types comme expliqué précédemment, et nous lisons les données dans le fichier `.dat`. Après avoir créé la série, nous utilisons `fn_export_ascii` pour exporter les résultats.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "chrlocal.h"
#include "chronos_msg.h"
#include "chronos_cmn.h"
#include "chronos_def.h"
#include "chronos_ext.h"

#define NDIM 4 /* total number of variables */

int main()
{

  int i,l;
  DOUBLE coef,cst,L1,L2;;

  INT2 k[NDIM];
  UNS2 perm[NDIM], count[NDIM], type[NDIM];
  INT2 smin[NDIM], smax[NDIM];

  int k1,k2,k3,k4;
  DMHDR *idxfr, *derfr, *prodfr, *genfr, *canfr;

  DMHDR *fdmh;
  const double pi=3.14159265358979323;
  double m0,m1,m2,a1,a2,G,cst,L1,L2;

  FN_HANDLERS fh;

   /* initialize Chronos */
  chr_init();


  m0=1.31;
  m1=1.92/1047.355;
  m2=4.13/1047.355;
  a1=0.832;
  a2=2.53;
  G=4*pow(pi,2);
  cst=-G*m1*m2/a2;
```

```
L1=m1*pow(G*m0*a1,0.5);
L2=m2*pow(G*m0*a2,0.5);


/* permutation vector */
for(i = 0; i < NDIM; ++i)
  perm[i] = i+1;
/* 2 fragment of dimension 2 */
count[0] = 2;
count[1]=2;
/* of fourier and polynomial type */
type[0]=IDX_FOURIER;
type[1] = IDX_POLYNOM;
/* with degree at least 0 */
smin[0]=0;
smin[1] = 0;
/* and at most 12 */
smax[0]=12;
smax[1] = 12;

/* say to Chronos "how to store" */
idxfr = fn_def_idx_fragm(NDIM, perm, 2, count, type, smin, smax, NULL, NULL);

/* the derivatives are.. */
type[0] = DER_FOURIER;
type[1] = DER_POLYNOM;
derfr = fn_def_fragm(NDIM, perm, 2, count, type);
/* the products are... */
type[0] = PROD_FOURIER;
type[1] = PROD_POLYNOM;
prodfr = fn_def_fragm(NDIM, perm, 2, count, type);
/* no rule on "how to compute the generating functions" */
type[0] = GEN_UNDEF;
type[1] = GEN_UNDEF;
genfr = fn_def_fragm(NDIM, perm, 2, count, type);
/*  canonical structure */
type[0] = CAN_R_GENERIC;
type[1] = CAN_R_GENERIC;
canfr = fn_def_fragm(NDIM, perm, 2, count, type);

/*  we creates the wanted Poisson series */
fdmh = fn_create(NDIM, 0, idxfr, derfr, prodfr, genfr, canfr, FN_BIN_TREE, CF_DOUBLE);

/* free the memory  */
dm_release(idxfr);
dm_release(derfr);
dm_release(prodfr);
dm_release(genfr);
dm_release(canfr);
printf("done.\n");

/*Read the data */
```

```
  get_fn_handlers(fdmh, &fh);
  {

    char xcf[fh.coefsize];
    fich=fopen("series.dat","r");

    while(fgetc(fich)!=EOF){
      for(i = 0; i < NDIM; k[i++] = 0);
      fscanf(fich,"%d %d %d %d %le",&k1,&k2,&k3,&k4,&coef);
      k[0]=k1;
      k[1]=k2;
      k[2]=k3;
      k[3]=k4;
      fh.cv_from[BCFTYP_DOUBLE](&coef, xcf);

      fh.add_coef(xcf, k, fdmh);
    }
    for(i = 0; i < NDIM; k[i++] = 0);
    fclose(fich);
  }

  /*multiply the coef by cst */
  get_fn_handlers(fdmh,&fh);
  CHR_ICF_TY cof[fh.coefsize];

  fh.cv_from[BCFTYP_DOUBLE](&cst,cof);
  scalar_multiply(fdmh,cof);

  /* Print result */
  printf("***** Hamiltonien \n");
  fn_printnz_coef(fdmh);

  /*Export result for further use */
  fn_export_ascii(fdmh,"myser.asc");

  /*Free memory */
  fn_remove(fdmh);


  /* some info about memory usage */
  dm_shuffle(0);
  db_list_summary();

  /* close Chronos */
  chr_finish();

  printf("Test completed.\n");

  return(0);
}
```

7.5 Conclusions

Suite aux exemples précédents, nous avons pu constater de nombreuses différences entre le MSNam et Chronos.

Chronos est bien plus complexe et plus général que le MSNam, mais n'oublions pas que c'est le choix de leur créateur respectif. De plus, cette complexité peut également jouer au désavantage de Chronos, le rendant beaucoup plus difficile d'accès.

Nous noterons également que Chronos est encore en évolution, Antonio Giorgilli travaillant encore et toujours sur de nouvelles améliorations.

Le MSNam peut certainement recevoir de nouvelles améliorations, notamment un usage dynamique de la mémoire, mais il jouit, entre autre grâce à ce travail, d'une accessibilité plus grande que Chronos.

Il existe certainement bien d'autres manières de comparer les manipulateurs, mais nous nous sommes restreints à décrire certaines différences flagrantes. Jacques Henrard et Antonio Giorgilli avaient émis l'hypothèse de prendre les meilleurs aspects de chaque manipulateur pour en obtenir un plus performant. Il semble à l'heure actuelle que cette tâche serait extrêmement compliquée, mais tout reste possible.

En conclusion de cette comparaison nous dirons que chaque manipulateur possède ses avantages et ses inconvénients, qu'ils ont été créés dans des buts différents et que chacun peut encore être amélioré.

Conclusion et perspectives

Tout au long de ce travail, nous avons œuvré à rendre le MSNam accessible à tous. En faisant un descriptif complet d'une part, et en ajoutant de nouvelles routines d'autre part, nous avons amélioré le manipulateur. Nous espérons que ce manuscrit rendra le MSNam plus attractif et attirera de nouveaux utilisateurs d'ici ou d'ailleurs.

Nous avons démontré lors d'exemples et d'applications, les nombreuses possibilités qu'offre le MSNam. Il a déjà fait ses preuves par le passé, pour son créateur notamment, mais également pour de nombreux utilisateurs au sein de notre départemment tels qu'Anne-Sophie Libert, Stéphane Valk ou Julien Dufey par exemple.

Bien que nous n'ayons pas fait d'avancée majeure ou de découverte importante pour la mécanique céleste dans ce travail, nous estimons qu'en rendant accessible un outil tel que celui-ci à des utilisateurs potentiels, nous participons indirectement à des découvertes futures.

Nous avons également pu remarquer qu'il reste, et qu'il y aura toujours des améliorations à apporter. De futurs travaux pourraient notamment porter sur une meilleure gestion de la mémoire, en utilisant des allocations dynamiques comme le fait Chronos par exemple. Certains penseront également à élargir les possibilités qu'offrent le MSNam en permettant l'utilisation d'autres types de données (variables complexes, polynômes, etc.). Mais n'oublions pas que le MSNam a été créé par et pour des gens qui travaillent en mécanique céleste.

Le MSNam a encore de beaux jours devant lui et si, grâce à ce travail, nous permettons la pérennité et l'expansion de cet outil, alors nous serons heureux d'y avoir participé.

Annexe A

Aide du MSNam par Jacques Henrard

SUMMARY OF THE MSNam

The first version in Fortran 77 is due to the late Michèle Moons;

This augmented version in Fortran 90 was coded by Jacques Henrard (July 2004).

INTRODUCTION

`MSNam` is a set of subroutines written in FORTAN90 for manipulating *Poisson series* of the form:

$$S = \sum_{\substack{i_1,\cdots,i_p \\ j_1\cdots,j_q}} A_{\substack{i_1,\cdots,i_p \\ j_1\cdots,j_q}} \; x_1^{i_1} \cdots x_p^{i_p} \left\{ \begin{matrix} \cos \\ \sin \end{matrix} \right\} (j_1\phi_1 + \cdots + j_q\phi_q) \; ,$$

i.e. Fourier sums in q angles $\phi_1, \cdots, \phi_q$ the coefficients of which are polynomials in p variables $x_1, \cdots, x_p$.

The *arguments* $(j_1, \cdots, j_p)$ and *exponents* $(i_1, \cdots, i_q)$ are integers and the *coefficients* $A_{\{\ldots\}}$ are double precision real numbers. The arguments and exponents and the indication that the trigonometric expression is a cosine or a sine are coded and packed in a large array of integers, the shape and length of which should be specified by the user (see `module parameters`); the coefficients $A_{\substack{i_1,\cdots,i_p \\ j_1\cdots,j_q}}$ are stored in a large array of double precision numbers.

Each series is identified by an integer which indicates the position of its first term in the large arrays. It is this identifier which is coded in the calling sequence of the subroutines. The first time a series is used its identifier should be set at zero.

THE MODULE "PARAMETERS"

The module `parameters` contains the parameters of the `MSNam` which should be defined by the user. It is assumed that the set of parameters in the module will be defined for a full research project, but of course the user can always change them in the course of the evolution of the research. In that case, special care should be given in the way the series are passed from one program to another one using a different set of parameters. Some hints to that effect will be given in the appendix HINTS.

The parameters are:

- `accuracy`: At the end of some subroutines, namely `ACUM`, `PROD`, `SCALE`, `SCALEP`, the terms of the series, the absolute value of which are less than `accuracy`, are dropped

in order to eliminate non significant terms. We recommend to set it at 10^{-14}, but of course the user may change it.

- `nwords`: number of words (32 bits integers) allowed to code the arguments and exponents of each term of a series. This number depends of course on the number of arguments and exponents and of the maximum values allowed for each of them (see below for the determination of the minimum number of words needed).
- `nvt`: number of trigonometric variables.
- `nvp`: number of polynomial variables.
- `nvar`: It is automatically set to the sum of `nvt` and `nvp`.
- `storelength`: Maximum number of terms in all the series present at a given time. The `MSNam` will reserve (in the module `tables`) an array `TABLE (nwords,storelength)` in order to store the code representing the arguments, exponents and the sine-cosine flag.
- `nmaxser`: maximum number of series present at the same time.
- `ordmax`: maximum order for the vectorial subroutines (see the section: vectorial subroutines).
- `nmax(nvar)`: indicates, for each argument and exponent, the maximum allowed for its absolute value. If `nmax(k)` if between 2^{n-1} and 2^n, the code for the variable `k` will take n bits. The module `parameters` offers, as a comment, a small table giving the largest absolute value which can be coded in a given number of bits. Each word of 32 bits will code an integer number of arguments or exponents (see the appendix); hence in each word some bits may be unused and reclaimed by allowing larger absolute values for some arguments or exponents.To help the user to make these choices and to check if the number of words allowed is sufficient, a program `test_parameters` can be run to see in which word the various arguments and exponents are coded and how many bits are unused in each word.
- `namevt(nvt)`: the names given to the trigonometric variables, coded each in a character string of length 4. These name are used in the printout of the series, but also, in somme subroutines (for instance `PDERT`) to identify the variable with respect to which some action (for instance a partial derivative) is taken.
- `namevp(nvp)`: the names given to the polynomial variables, coded each in a character string of length 4. These names are used in the printout of the series, but also, in the subroutine `PDERP`, `XMON`, `SCALEP`, to identify the variable with respect to which some action (derivative, multiplication, scaling) is taken.

THE MODULE "TABLES"

This module defines the tables to be used to code the series and several parameters needed for the working of the subroutines. Both modules are needed for the working of the subroutines of `MSNam`.

CONVENTIONS

- Variables in capital latin letters stand for series identificators (defined as 32-bits integers).
- Greek letters stand for double precision real numbers.
- Variables in lower case latin letters stand for integer numbers.
- Variables in italic stand for string of characters.

INITIALIZATION SUBROUTINE

`START_MS` : this subroutine should be called before any other subroutines of `MSNam`. It initialize several quantities needed for the working of the other subroutines.

SUBROUTINE FOR BUILDING AND DISPLAYING SERIES

`STORE(A,sc,arg,exp,`α`)` : add to the series `A` a term the sine-cosine flag of which is given in `sc`, the arguments and exponents of which are coded in `arg` and `exp`, and the numerical coefficient of which is α.

NOTE: the dimension of arg (resp. exp) should be nvt (resp. nvp) or longer. The integer sc is 0 for a cosine and 1 for a sine.

`CONSTANT (A,`α`)`: subroutine `CONSTANT` creates a series `A` with one term. The sc-flag, trigonometric and polynomial arguments are set to zero and the numerical coefficient to α.

`NBTERM (A)`: the integer function `NBTERM` gives the number of terms of the series identified by `A`.

`FETCH(A,j,sc,arg,exp,`α`)`: the subroutine `FETCH` decodes the j-th term of the series A; the sc-flag is placed in `sc`, the trigonometric (resp. polynomial) arguments in the vector `arg` (resp. `exp`) and the numerical coefficient in α. The subroutine is usually called within a do loop from 1 to `NBTERM(A)` in order to operate on each terms of the series.

NOTE: the dimension of arg (resp. exp) should be nvt (resp. nvp) or longer. The

integer sc is 0 for a cosine and 1 for a sine.

`PRINT (io,A,`*label*`,i)` : the subroutine `PRINT` output in the FORTRAN file `io` the series `A` under an identifying label composed of the character string *label* and the integer i. The output consist of a list of the terms under the form (sine-cosine flag, arguments, exponents, coefficient).

NOTE: see `ORDER_SIZE`, below for printing with a special ordering of the terms.

`READ (io,A)` : the subroutine `READ` inputs the series `A` from the unit designated by `io`. The unit is assumed to be `open` for input. The subroutine assumes that the file has been constructed by the subroutine `PRINT`. The subroutine checks that the current names of the variables match the names attributed when forming the file. The series `A` should be an empty series.

SUBROUTINE FOR SERIES MANAGEMENT

`COPY (A,B)`: the subroutine `COPY` takes a copy of the series `A` and identifies it by B. The identifier `B` should be zero (identifier of an empty series) before the call.

`ERASE (A)`: the subroutine `ERASE` destroys the series `A` and reclaims the room previously occupied by it.

`RENAME (A,B)` : the subroutine `RENAME` exchanges the identifiers of the series `A` and `B`.

`CUTEPS (A,`ϵ`)`: the subroutine `CUTEPS` drops, in the series identified by `A`, all the terms with a coefficient less (in absolute value) than ϵ.

`CHANGE_NAME(`*oldname, newname*`)` : the subroutines substitute to the name `oldname` of a variable the new name `newname`. This is used mainly before and after the call to a specialized routine (like `dev2B_fr` – see below) which expects variables with a specific name; or before printing if some variables have changed meaning in the course of the program.

`INDEXARG(`*name*`)` : integer function the value of which is the rank of the trigonometric variable the name of which is *name*.

`INDEXPOL(`*name*`)` : integer function the value of which is the rank of the polynomial variable the name of which is *name*.

`ORDER_SIZE(ser)` : the subroutines order the series according to the size of the absolute value of the coefficients. This ordering is necessary for the arguments `A` and `B` of `PRODCT` and this subroutine will call it. The subroutine may also be useful before a `PRINT`, if the user wants the terms to be printed in that order. In that case it is the responsability of the user to call the ordering subroutine.

NOTE: The routine changes the sign of the identifier of the series ordered this way so that other subroutines know about the ordering.

`ORDER_CODE(ser)` : the subroutines order the series according to the lexographic order of the keys. This ordering is necessary for series which can possibly be changed (which are output of such operations as `STORE`, `ACUM`, `PROD`, `PRODCT`,`PDERT`, `PDERP`). These subroutines will automatically call the ordering subroutine if need be.

SUBROUTINE FOR ALGEBRAIC OPERATIONS

`SCALE (A,`α`)`: the subroutine `SCALE` multiplies the series identified by `A` by α.

`SCALEP (A,` *namepol*,α`)`: the subroutine `SCALEP` scales, in the series identified by `A`, the polynomial variable identified by *namepol* by the factor α.

`EVALP (A,` *namepol*,α`)`: in the series identified by `A`, the subroutine `EVALP` gives the value α to the polynomial variable identified by *namepol*.

`EVALT (A,` *nametrig*,α`)`: in the series identified by `A`, the subroutine `EVALT` gives the value α to the trigonometric variable identified by *nametrig*.

`XMON (A,`*namepol*,`powr)`: the subroutine `XMON` multiply the series identified by `A` by the monomial variable identified by *namepol* to the power `powr`.

`ACUM (A,B,`α`)`: the subroutine `ACUM` replaces the series `B` by its previous content augmented by the product α*`A`.

`PROD (A,B,C,`α`)`: the subroutine `PROD` replaces the series `C` by its previous content augmented by the product α*`A`*`B`.

`PRODCT (A,B,C,`α`,` ϵ`)`: the subroutine `PRODCT` acts as the subroutine `PROD`, except that the product of individual terms is discarded each time the absolute value of its coefficient is smaller than ϵ. The fact that `A` and `B` are ordered according to size enables the subroutine to skip many products of individual terms without even considering them. Considering a possible accumulation of partial results, it is advised to set the level of truncature ϵ well below (a factor of 10?) the overall level of truncature set in the subroutine `CUTEPS` which it is recommended to call after the product.

`PDERT (A,B,`*nametrig*`)`: the subroutine `PDERT` add to the series `B` the derivative of `A` with respect to the trigonometric variable identified by *nametrig*.

`PDERP (A,B,`*namepol*`)`: the subroutine `PDERP` add to the series `B` the derivative of `A` with respect to the polynomial variable identified by *namepol*.

`SUBSTITUTE (A,B,`*namepol*`)`: the subroutine `SUBSTITUTE` replace in the series `A` the variable *namepol* by the series `B`. The variable *namepol* should not appear to a negative power in the series `A`.

VECTORIAL OPERATIONS

The vectorial operations assume that the "series" coded in the calling sequence (like `A,B,C` below) are actually arrays of the type `A(0:order)`. The results are computed only up to `order`. The value of `order` should be smaller or equal to `ordmax` defined in the module `parameters`. For instance for the product $A * B \rightarrow C$, the routines computes :

$$
\begin{aligned}
C(0) &= C(0) + A(0) * B(0) \\
C(1) &= C(1) + A(0) * B(1) + A(1) * B(0) \\
C(2) &= C(2) + A(0) * B(2) + 2 * A(1) * B(1) + A(2) * B(0) \\
&\cdots
\end{aligned}
$$

`V_COPY(A,B,order)`

`V_ERASE(A,order)`

`V_RENAME(A,B,order)`

`V_CUTEPS(A,`ϵ`,order)`

`V_PRINT(io,A,`*label*`,order)`

`V_SCALE(A,`α`,order)`

`V_ACUM(A,B,`α`,order)`

`V_PROD(A,B,C,`α`,order)`

`V_SUBSTITUTE (A,B,`*namepol*`,order)`: both `A` and `B` are vectors: `A(0:order)` and `B(0:order)`. The subroutine `V_SUBSTITUTE` replace in the series `A(n)` the variable *namepol* by the expansion up to order `order` of the series `B=`$\sum_k$ `B(k)`. The variable *namepol* should not appear to a negative power in the series `A(n)`.

NOTE: A frequent error of users is to forget to code the argument `order`.

LOW LEVEL SUBROUTINES : They should not be called by the "normal user". They are described here for the sake of completeness.

- `INIT(A)` : Attribute a location to a new series named when `A`, when its first term is created. It is called by the subroutines `COPY` and `PUTTRM`.
- `MOVEBLOCK(lg,oldbeg,newbeg)` : moves blocks of data in the tables. It is called by the subroutines `MOVE_TO_END` and `ERASE`.
- `MOVE_TO_END(A)` : moves the series `A` to the end of the working area. It is called by the subroutine `PUTTRM`.

- ENCODE(code,sc,arg,exp) : codes in the vector code, the sine-cosine flag sc and the arguments arg and exponents exp. It is called by the subroutine STORE.
- DECODE(code,sc,arg,exp) : reverse operation; decodes from the vector code, the sine-cosine flag sc and the arguments arg and exponents exp. It is called by the subroutines FETCH and SCALEP.
- SEARCH (A,code,left): the subroutine SEARCH indicates in left the location of the term identified by the coded array code in the series A , or, if such a term does not exist in A, the location of the term which should immediately preceed it. The subroutine is called by PUTTRM.
- PUTTRM(A,code,α) : the subroutine adds to the series A, a term the arguments and exponents of which are coded in code, and the numerical coefficient of which is α. It is called by the subroutines STORE and ACUM.
- ERROR : handles the error messages of the other subroutines of MSNam. It is called by most of the subroutines.

SPECIALIZED OPERATIONS

POWER(A,α,order)

The subroutine POWER assumes that A is an array A(0:order), with A(k) the component of order k in an expansion in powers of some "small parameter". As an input, the series A(0) is supposed to be the unit series (1.*cos(0)). The subroutine raises A to the power α, up to order order in powers of the small parameter.

POLAR_TO_CART(A,*nameR*,*nameT*,*nameX*,*nameY*)

The variables *nameR* and *nameT* are the polar coordinates (*nameR* the distance and *nameT* the angle) of a point of the plane, and the polynomial variables *nameX* and *nameY* the cartesian coordinates of the same point. The subroutine POLAR_TO_CART transforms, in the series A, the polar coordinates into cartesian coordinates.

CART_TO_POLAR(A,*nameR*,*nameT*,*nameX*,*nameY*)

The polynomial variables *nameX* and *nameY* are the cartesian coordinates of a point of the plane and the variables *nameR* and *nameT* the polar coordinates (*nameR* the distance and *nameT* the angle) of the same point. The subroutine CART_TO_POLAR transforms, in the series A, the cartesian coordinates in polar coordinates.

`DEV2B_FR(A,order)`

The subroutine assumes that two polynomial variables are identified respectively by ' r' and ' e' and one trigonometric variables by ' f'. The argument `A` is an array `A(0:order)`, where `A(k)` is the component of order `k` in the expansion in powers of ' e'. In the frame of the two body problem, ' r' stands for the normalised distance r/a, ' e' for the eccentricity and ' f' for the true anomaly. The subroutine replaces the function `A` by its expansion in terms of the eccentricity and the mean anomaly. The result is independent of the variable ' r' (i.e. all the exponents of ' r' are set to zero). In the output `A`, the variable ' f' designates the mean anomaly.

Appendix A: Packing the arguments and exponents

The arguments $[j_1, \cdots, j_q]$ and the exponents $[i_1, \cdots, i_p]$ of each term of a series are packed in an array of integers. First the two arrays of arguments and exponents are concatenated in one array $[k(1) \cdots k(q+p)]$, first the arguments then the exponents in the same order. Each integer $k(\ell)$ in the k-list has been assigned a number of bits $n(\ell)$ by the module `parameters`. Starting from the first, each elements of the array of coded words packs m elements of the k-list, with m defined in such a way that the number of bits necessary to code them is less or equal to 31.

Appendix B: Hints

Annexe B

Le MSNam

```
! ************************************************************!
!  Welcome in the MSNam, for help consult file MSNam-help.pdf  !
!                                                              !
!  Created by Michele Moons                                    !
!  Modified in 2004 by Jacques Henrard                         !
!  Latest update in 2012 by Joffroy Henrard, 2012 -2013        !
!                                                              !
!************************************************************!


!*********************************************************************!
! Module MSparameters                                                 !
! This module provides the sizes of all parameters needed in the MSNam !
! NOTE : update the values if needed before compiling                 !
!                                                                     !
!*********************************************************************!

module MSparameters
  implicit none
  save
  integer, parameter ::dp=kind(1d0)
  real(kind=dp),parameter :: MSaccuracy=1.d-13
  integer, parameter ::MSnwords=5, MSnvt=6,MSnvp=12,MSnvar=MSnvt+MSnvp
  integer, parameter ::MSstorelength=600000, MSnmaxser=200
  integer, parameter ::MSordmax=20

  integer, dimension(MSnvar),parameter:: MSnmax =    &
       & (/31,31,31,31,31,31,31,31,31,31,31,31,31,31,31,31,31,31,31/)

  !  MSnmax=3 -> 2 bits ; MSnmax=7 -> 3 bits ; MSnmax=15 -> 4 bits
  !  MSnmax=31 -> 5 bits ; MSnmax=63 -> 6 bits ; . . . . .

  character(len=4), dimension(MSnvt) :: MSnamevt=  &
       &(/' l1',' l2',' om1',' om2', ' Om1',' Om2'/)
  character(len=4), dimension(MSnvp) :: MSnamevp=      &
       &(/'  S1','  S2','  E1','  E2', ' sL1', ' sL2', '  X1', '  X2', ' xi1',' xi2','eta1','eta2'/)

end module MSparameters
```

```
!*********************************************************************!
! Module MSTABLES                                                     !
! This module creates the variables needed to use the MSNam,          !
! using the sizes defined earlier in MSparameters                     !
!                                                                     !
!*********************************************************************!

module MSTABLES
  use MSparameters
  implicit none

  real(kind=dp),save:: MStabcoef(MSstorelength)
  integer,save:: MSTABLE(MSnwords,MSstorelength)
  integer,save:: MSfree,MSnser,MSbegin(MSnmaxser),MSlength(MSnmaxser)
  integer,save:: MSshift(MSnvar),MSnvw(MSnwords)
  character(len=50),save::MSformprint,MSformread
  character(len=4),save:: MSheader(MSnvar+7)
  integer:: MSintmess(3)=(/0,0,0/),MStrigpri(MSnvt),MSexppri(MSnvp)
  character(len=80):: MSmessage
  character(len=4):: MScharmess(2)=(/'    ','    '/)

end module MSTABLES




!*********************************************************************!
! subroutine INIT(ser)                                                !
! The subroutine INIT attributes a location to a new series named ser, !
! when its first term is created.                                     !
!                                                                     !
!*********************************************************************!
subroutine INIT(ser)
  use MSTABLES
  implicit none
  integer,intent(inout):: ser
  integer:: k

  ! Validation

  if(ser.ne.0) then   ! if the series already exists
     MSmessage='error in INIT: identifier = (1) '
     MSintmess=0
     MSintmess(1)=ser
     call ERROR
  endif

  ! Seach for a free identifier of series
```

```
  do k=1,MSnmaxser
     if(MSbegin(k).eq.0) goto 10
  enddo
  MSmessage='error in INIT: too many series'
  call ERROR


10 ser=k

  ! MSnser is the number of active series

  if(k.gt.MSnser) MSnser=k

  MSbegin(ser)=MSfree
  MSlength(ser)=0

end subroutine INIT


!*********************************************************************!
! subroutine MOVEBLOCK(lg,oldbeg,newbeg)                              !
! The subroutine MOVEBLOCK moves blocks of data in the tables         !
!                                                                     !
!*********************************************************************!
subroutine MOVEBLOCK(lg,oldbeg,newbeg)
  use MSTABLES
  implicit none
  integer,intent(in):: lg,oldbeg,newbeg
  integer:: j,k

  ! Validation
  ! Check if we can move the data
  if(oldbeg.lt.newbeg.and.newbeg.ne.MSfree) then
     MSmessage='error in MOVEBLOCK: some data might be written over &
          &oldbeg=(1),newbeg=(2),MSfree =(3). '
     MSintmess=0
     MSintmess(1)=oldbeg
     MSintmess(2)=newbeg
     MSintmess(3)=MSfree
     call ERROR

     if((newbeg+lg).gt.MSstorelength) then
        MSmessage='error in MOVEBLOCK: no room left. MSfree=(1)&
             &,lg=(2),last=(3)'
        MSintmess=0
        MSintmess(1)=MSfree
        MSintmess(2)=lg
        MSintmess(3)=MSstorelength
        call ERROR
     endif
  endif
```

```
  ! Move the data

  do k=0,lg-1
     do j=1,MSnwords
        MSTABLE(j,newbeg+k)=MSTABLE(j,oldbeg+k)
     enddo

     MStabcoef(newbeg+k)=MStabcoef(oldbeg+k)
  enddo

end subroutine MOVEBLOCK


!*********************************************************************!
! subroutine MOVE_TO_END(ser)                                         !
! The subroutine INIT moves the series ser to the end of the working  !
! area                                                                !
!                                                                     !
!*********************************************************************!
subroutine MOVE_TO_END(ser)
  use MSTABLES
  implicit none
  integer,intent(in):: ser
  integer:: k,begser,lgser,endser

  ! Validation
  if(ser.eq.0) return   !if the series is empty
  if(abs(ser).gt.MSnmaxser) then  !if there are more series than nmaxser
     MSmessage='error in MOVE_TO_END: the identifier (1) is&
          & not valid '
     MSintmess=0
     MSintmess(1)=ser
     call ERROR
  endif

  begser=MSbegin(abs(ser))
  lgser=MSlength(abs(ser))
  endser=begser+lgser

  ! The series is already at the end

  if(endser.eq.MSfree) return

  ! Move the series

  call MOVEBLOCK(lgser,begser,MSfree)

  ! Reclaim the room previously occupied by the series

  call MOVEBLOCK(MSfree-begser,endser,begser)
```

```
  ! Adjust the beginning of the series which have been moved

  do k=1,MSnmaxser
     if(MSbegin(k).gt.begser) MSbegin(k)=MSbegin(k)-lgser
  enddo
  MSbegin(ser)=MSfree-lgser

end subroutine MOVE_TO_END


!**********************************************************************!
! subroutine SEARCH(ser,arg,left)                                      !
! The subroutine SEARCH  indicates in left the location of the term    !
! identified by the coded array code in the series A , or, if such a   !
! term does not exist in A, the location of the term which should      !
! immediately preceed it.                                              !
!                                                                      !
!**********************************************************************!
subroutine SEARCH(ser,arg,left)
  use MSTABLES
  implicit none
  integer,intent(in)::ser,arg(MSnwords)
  integer,intent(out)::left
  integer:: k,right,middle,dif

  ! validation

  if(abs(ser).gt.MSnmaxser) then !if there are more series than nmaxser
     MSmessage='error in SEARCH: the identifier (1) is&
          & not valid '
     MSintmess=0
     MSintmess(1)=ser
     call ERROR
  endif

  ! Order in lexicographic order

  if(ser.lt.0) call ORDER_CODE(ser)


  left=MSbegin(ser)-1
  right=MSbegin(ser)+MSlength(ser)

10 if(right.eq.left+1) return

  ! Start the dichotomy

  middle=(right+left)/2
  do k=1,MSnwords
     dif=MSTABLE(k,middle)-arg(k)
     if(dif.lt.0) then
        ! dif< 0 the terms is to the right of "middle"
```

```
        left=middle
        goto 10
      endif
      if(dif.gt.0) then
        ! dif> 0 the terms is to the left of "middle"
        right=middle
        goto 10
      endif
   enddo

   ! dif=0 for all words -> the term exists and is at "middle"
   left=middle

end subroutine SEARCH


!*********************************************************************!
! subroutine PUTTRM(ser,arg,coefj)                                    !
! The subroutine PUTTRM adds to the series ser, a term the arguments  !
! and exponents of which are coded in arg, and the numerical          !
! coefficient of which is coefj                                       !
!                                                                     !
!*********************************************************************!
subroutine PUTTRM(ser,arg,coefj)
  use MSTABLES
  implicit none
  double precision,intent(in)::coefj
  integer,intent(in):: arg(MSnwords)
  integer,intent(inout)::ser
  integer:: j,k,place,lg,knew,left
  logical::same

  ! validation

  if(abs(ser).gt.MSnmaxser) then !if there are more series than nmaxser
     MSmessage='error in PUTTRM: the identifier (1) is&
          & not valid '
     MSintmess=0
     MSintmess(1)=ser
     call ERROR
  endif

  ! Order in lexicographic order

  if(ser.lt.0) call ORDER_CODE(ser)

  ! if the series does not exist, define it

  if(ser.eq.0) then
     call INIT(ser)
     place=MSfree
```

```
     !  Otherwise, search for a similar term

  else
     call SEARCH(ser,arg,left)
     same=.false.
     if(left.ne.(MSbegin(ser)-1)) then
        same=.true.
        do k=1,MSnwords
           if(arg(k).ne.MSTABLE(k,left)) same=.false.
        enddo
     endif

     if(same) then
        ! it exists already
        MStabcoef(left)=MStabcoef(left)+coefj
        return
     else
        ! it is new, the stack has to be moved to make room
        lg=MSbegin(ser)+MSlength(ser)-left-1
        call MOVE_TO_END(ser)
        if(MSfree.eq.MSstorelength) then
           MSmessage='no room left in stack: MSfree (1) = MSstorelength&
                & (2)'
           MSintmess(1)=MSfree
           MSintmess(2)=MSstorelength
           call ERROR
        endif
        do k=MSfree-1,MSfree-lg,-1
           knew=k+1
           do j=1,MSnwords
              MSTABLE(j,knew)=MSTABLE(j,k)
           enddo
           MStabcoef(knew)=MStabcoef(k)

        enddo

        place=MSfree-lg
     endif
  endif
  ! The new term has to be inserted
  do k=1,MSnwords
     MSTABLE(k,place)=arg(k)
  enddo
  MStabcoef(place)=coefj
  MSlength(ser)=MSlength(ser)+1
  MSfree=MSfree+1

end subroutine PUTTRM

!*********************************************************************!
! subroutine ENCODE(code,sc,arg,exp)                                  !
! The subroutine ENCODE codes in the vector code, the sine-cosine flag !
```

```
! sc, the arguments arg and exponents exp                                !
!                                                                        !
!************************************************************************!
subroutine ENCODE(code,sc,arg,exp)
  use MSTABLES
  implicit none
  integer,intent(in):: sc,arg(MSnvt),exp(MSnvp)
  integer,intent(out):: code(MSnwords)
  integer:: j,k,tablearg(MSnvar),sign,start

  ! concatenate the arguments

  tablearg(1:MSnvt)=arg
  tablearg(MSnvt+1:MSnvar)=exp

  ! Validation
  ! test if the arguments are within their ranges

  do k=1,MSnvar
     if(abs(tablearg(k)).gt.MSnmax(k)) then
        MSmessage='error in ENCODE: argument (1) is equal&
             & to (2) and out of range'
        MSintmess=0
        MSintmess(1)=k
        MSintmess(2)=tablearg(k)
        call ERROR
     endif
  enddo

  ! coding

  start=0
  do k=1,MSnwords
     code(k)=0

     do j=start+1,start+MSnvw(k)
        if(tablearg(j).lt.0) then
           sign=1
        else
           sign=0
        endif
        code(k)=2*code(k)+sign
        code(k)=ishft(code(k),MSshift(j))
        code(k)=code(k)+abs(tablearg(j))
     enddo
     start=start+MSnvw(k)
  enddo
  code(1)=2*code(1)+sc

end subroutine ENCODE

!************************************************************************!
```

```
! subroutine DECODE(code,sc,arg,exp)                                    !
! The subroutine DECODE decodes from the vector code, the sine-cosine  !
! flag sc, the arguments arg and exponents exp                         !
!                                                                      !
!**********************************************************************!
subroutine DECODE(code,sc,arg,exp)
  use MSTABLES
  implicit none
  integer,intent(in):: code(MSnwords)
  integer,intent(out):: sc,arg(MSnvt),exp(MSnvp)
  integer:: j,k,tablearg(MSnvar),sign,start,shiftarg,temp1,temp2

  ! decode the sc flag
  sc=mod(code(1),2)

  ! decode the arguments

  start=0
  do k=1,MSnwords
     temp1=code(k)
     if(k.eq.1) temp1=temp1/2
     do j=start+MSnvw(k),start+1,-1
        shiftarg=ishft(temp1,-MSshift(j))
        sign=mod(shiftarg,2)
        temp2=ishft(shiftarg,MSshift(j))
        tablearg(j)=temp1-temp2
        if(sign.eq.1) tablearg(j)=-tablearg(j)
        temp1=shiftarg/2
     enddo
     start=start+MSnvw(k)
  enddo

  ! separate trig and poly

  arg=tablearg(1:MSnvt)
  exp=tablearg(MSnvt+1:MSnvar)

end subroutine DECODE

!**********************************************************************!
! subroutine ERROR                                                     !
! The subroutine ERROR handles the error messages of the other         !
! subroutines of the MSNam                                             !
!                                                                      !
!**********************************************************************!
subroutine ERROR
  use MSTABLES

  open (unit=33,file='error_log',status='new')
  write(33,*) MSmessage
  write(33,101) MSintmess
  write(33,102) MScharmess
```

```
101 format(' integer (1) = ',I8,' , (2) = ',I8,' , (3) = ',I8)
102 format(' character (a) = *',A4,'* , (b) = *',A4,'*')
  close (unit=33)

  stop
end subroutine ERROR

!**********************************************************************!
! subroutine START_MS                                                  !
! The subroutine START_MS should be called before any routine of the   !
! MSNam. It initialize several quantities needed for the working of    !
! the other subroutines                                                !
!                                                                      !
!**********************************************************************!
subroutine START_MS
  use MSTABLES
  implicit none
  integer:: j,k,it1,it2,it3,it4,bits,ncoded,iword,aux
  character(len=*),parameter::               &
       &Pform1= '(5x,A3,"(",'        ,    &
       &Pform2='I4," ) (",'          ,    &
       &Pform3='I4,")   " ,D23.16)'  ,    &
       &Rform1='(5x,A4,'             ,    &
       &Rform2='I4,4x,'              ,    &
       &Rform3='I4,4x,D23.16)'
  character(len=1),dimension(0:9):: &
       &digit=(/'0','1','2','3','4','5','6','7','8','9'/)
  character(len=4):: blank='    '

  ! Validation
  if(MSnvt.lt.0) then  ! if there are no trigonometric variables
     MSmessage='error in START_MS: MSnvt = '
     MSintmess=0
     MSintmess(1)=MSnvt
     call ERROR
  endif
  if(MSnvp.lt.0) then  ! if there are no polynomial variables
     MSmessage='error in START_MS: MSnvp. = '
     MSintmess=0
     MSintmess(1)=MSnvp
     call ERROR
  endif

  do k=1,MSnvar ! If there are no bit allowed to code
     if(MSnmax(k).lt.1) then ! if
        MSmessage='error in START_MS: the maximum &
             &for variable (1) is (2)'
        MSintmess=0
        MSintmess(1)=k
        MSintmess(2)=MSnmax(k)
        call ERROR
     endif
```

```
enddo

! Build the formats to be used for printing and reading
!  form the digits of MSnvt and MSnvp
it1=MSnvt/10
it2=MSnvt-10*(MSnvt/10)
it3=MSnvp/10
it4=MSnvp-10*(MSnvp/10)

MSformprint=Pform1//digit(it1)//digit(it2)            &
     &//Pform2//digit(it3)//digit(it4)//Pform3
MSformread=Rform1//digit(it1)//digit(it2)             &
     &//Rform2//digit(it3)//digit(it4)//Rform3
MSheader(1:2)=(/blank,blank/)
MSheader(3:MSnvt+2)=MSnamevt(1:MSnvt)
MSheader(MSnvt+3)=blank
MSheader(MSnvt+4:MSnvar+3)=MSnamevp(1:MSnvp)
MSheader(MSnvar+4:MSnvar+7)=(/blank,blank,blank,'COEF'/)

! Set the pointers to the series to zero

MSbegin=0
MSlength=0
MSnser=0
MSfree=1

! Compute the locations of the variables in the packed words

MSnvw=0
bits=1
iword=1

MSnvw(1)=MSnvar
ncoded=0
do k=1,MSnvar
   MSshift(k)=0
   aux=MSnmax(k)
   do j=1,30
      if(aux.eq.0) exit
      MSshift(k)=MSshift(k)+1
      aux=aux/2
   enddo

   bits=bits+MSshift(k)+1
   ! If there are not enough words to code the arguments
   if(bits.ge.32) then
      if(iword.eq.MSnwords) then
         MSmessage='error in START_MS: need more words&
              & to code the arguments. (1) nb of arg &
              &not coded'
         MSintmess=0
         MSintmess(1)=MSnvar-k+1
```

```
          call ERROR
        endif
        MSnvw(iword)=k-1-ncoded
        ncoded=k-1
        iword=iword+1
        MSnvw(iword)=MSnvar-ncoded
        bits=MSshift(k)+1

     endif
  enddo

end subroutine START_MS


!*********************************************************************!
! subroutine STORE(ser,sc,arg,exp,coef)                               !
! The subroutine STORE add to the series "ser" a term the sine-cosine  !
! flag of which is given in sc, arguments and exponents of which are   !
! coded in arg and  exp and the numerical coefficient of which is coef !
!                                                                     !
!*********************************************************************!
subroutine STORE(ser,sc,arg,exp,coef)
  use MSTABLES
  implicit none
  double precision,intent(in)::coef
  double precision::coefj
  integer,intent(in)::sc,arg(MSnvt),exp(MSnvp)
  integer,intent(inout)::ser
  integer:: k,argbis(MSnvt),code(MSnvar)

  ! Validation

  ! check that START_MS has been called
  if(MSheader(MSnvar+7).ne.'COEF')  then
     MSmessage='error in STORE: the subroutine START_MS &
          & has not been called '
     call ERROR
  endif

  ! check that "ser" is a valid identifier

  if(abs(ser).gt.MSnmaxser) then !if there are more series than nmaxser
     MSmessage='error in STORE: the identifier (1) is&
          & not valid '
     MSintmess=0
     MSintmess(1)=ser
     call ERROR
  endif

  ! Order in lexicographic order

  if(ser.lt.0) call ORDER_CODE(ser)
```

```
  ! make the first argument to be positive

  argbis=arg
  coefj=coef

  do k=1,MSnvt
     if(argbis(k).ne.0) then
        if(argbis(k).lt.0) then
           argbis=-argbis
           if(sc.eq.1) coefj=-coefj
        endif
        goto 10
     endif
  enddo

  ! The trig arguments are zero: return if it is a sine
  if(sc.eq.1) return
  ! otherwise
10 call ENCODE(code,sc,argbis,exp)

  ! If the series is ordered by size, order it by code

  if(ser.lt.0) then
     ser=-ser
     call ORDER_CODE(ser)
  endif

  call PUTTRM(ser,code,coefj)

end subroutine STORE


!**********************************************************************!
! subroutine CONSTANT(ser,alpha)                                       !
! The subroutine CONSTANT creates a series ser with one term.          !
! The sc-flag, trigonometric and polynomial arguments are set to 0     !
! and the numerical coefficient to alpha                               !
!                                                                      !
!**********************************************************************!
subroutine CONSTANT(ser,alpha)
  use MSTABLES
  implicit none
  double precision,intent(in)::alpha
  integer,intent(in)::ser
  integer:: sc,arg(MSnvt),exp(MSnvp)
  data sc/0/,arg/MSnvt*0/,exp/MSnvp*0/

  ! Validation
  if(ser.ne.0) then ! if the series already exists
     MSmessage='error in CONSTANT: the identifier (1) is&
          & not zero '
```

```
     MSintmess=0
     MSintmess(1)=ser
     call ERROR
  endif

  ! Form the series
  call STORE(ser,sc,arg,exp,alpha)

end subroutine CONSTANT


!*********************************************************************!
! function NBTERM(ser)                                                !
! The integer function NBTERM gives the number of terms of the series !
! identified by ser                                                   !
!                                                                     !
!*********************************************************************!
function NBTERM(ser)
  use MSTABLES
  implicit none
  integer,intent(in):: ser
  integer::NBTERM,absser

  ! Validation
  absser=abs(ser)
  if(absser.gt.MSnmaxser) then ! if there are more series than permitted
     MSmessage='error in NBTERM: the identifier (1) is&
          & not valid '
     MSintmess=0
     MSintmess(1)=ser
     call ERROR
  endif

  ! number of terms

  if(ser.eq.0) then ! if the series is empty
     NBTERM=0
  else
     NBTERM=MSlength(absser)
  endif

end function NBTERM


!*********************************************************************!
! subroutine FETCH(ser,j,sc,arg,exp,coef)                             !
! The subroutine FETCH decodes the j-th term of the series ser        !
! The sc-flag is placed in sc, the trigonometric and polynomial       !
! arguments respectively in arg and exp and the numerical coefficient !
! in coef.                                                            !
!                                                                     !
!*********************************************************************!
```

```
subroutine FETCH(ser,j,sc,arg,exp,coef)
  use MSTABLES
  implicit none
  double precision,intent(out)::coef
  integer,intent(in):: ser,j
  integer,intent(out)::sc,arg(MSnvt),exp(MSnvp)
  integer term,absser

  ! Validation
  absser=abs(ser)
  if(absser.gt.MSnmaxser) then ! if there are more series than permitted
     MSmessage='error in FETCH: the identifier (1) is&
          & not valid or zero'
     MSintmess=0
     MSintmess(1)=ser
     call ERROR
  endif

    ! if j is outside the range of the series
  if(j.lt.1.or.j.gt.MSlength(absser)) then
     MSmessage='error in FETCH: the index (1) is&
          & not between 1 and MSlength =(2) '
     MSintmess=0
     MSintmess(1)=j
     MSintmess(2)=MSlength(absser)
     call ERROR
  endif

  ! Decode the term

  term=j+MSbegin(absser)-1
  call DECODE(MSTABLE(1,term),sc,arg,exp)

  coef=MStabcoef(term)
end subroutine FETCH


!*********************************************************************!
! subroutine PRINT(IO,ser,label,int)                                  !
! The subroutine PRINT output in the FORTRAN file IO the series ser   !
! under an identifying label composed of the caracter string label and !
! the integer int. The output consist in a list of the terms under the !
! form (sc-flag, arguments, exponents, coefficient)                   !
!                                                                     !
!*********************************************************************!
subroutine PRINT(IO,ser,label,int)
  use MSTABLES
  implicit none
  double precision::coef
  integer,intent(in):: ser,IO,int
  integer k,sc,arg(MSnvt),exp(MSnvp),sign,NBTERM,nterm,absser
  character(len=*) ,intent(in):: label
```

```
  character(len=3),parameter :: cc='cos',cs='sin'
  ! ****************************************************************
  !  "IO" is an integer, unit number of a file open for write
  !  "ser" is the identifier if the series to be printed
  !  "label" is a character string to be printed as the label of
  !  the series
  !  "int" is an integer to be printed along the label
  ! ****************************************************************

  ! Validation

  if(abs(ser).gt.MSnmaxser) then ! if there are more series than permitted
     MSmessage='error in PRINT: the identifier (1) of the &
          &series to be printed is not valid '
     MSintmess=0
     MSintmess(1)=ser
     call ERROR
  endif



  ! number of terms

  nterm=NBTERM(ser)

  ! write the heading

  write(IO,1100)label,int,nterm
1100 format(///,10x,'SERIES',5X,A,5X,I5,/,10X,' NUMBER OF TERMS : '&
       &,I6,//)

  if(nterm.eq.0) return

  ! update the names of variables, and print them

  MSheader(3:MSnvt+2)=MSnamevt(1:MSnvt)
  MSheader(MSnvt+4:MSnvar+3)=MSnamevp(1:MSnvp)
  write(IO,1200) MSheader
1200 format(x,37A4)

  ! print the terms

  do k=1,nterm
     call FETCH(ser,k,sc,arg,exp,coef)
     if(sc.eq.0) then
        write(IO,MSformprint) cc,arg,exp,coef
     else
        write(IO,MSformprint) cs,arg,exp,coef
     endif
  enddo

end subroutine PRINT
```

```
!*********************************************************************!
! subroutine READ(IO,ser)                                             !
! The subroutine READ inputs the series ser from the unit designated  !
! by IO, which is supposed to be open for input and built by the      !
! subroutine PRINT.                                                   !
! The subroutine also checks that the current names of the variables  !
! match the names attributed when forming the file.                   !
!                                                                     !
!*********************************************************************!
subroutine READ(IO,ser)
  use MSTABLES
  implicit none
  double precision::coef
  integer,intent(in):: IO
  integer,intent(inout):: ser
  integer i,k,sc,arg(MSnvt),exp(MSnvp),sign,nterm
  character(len=29) :: file,number
  character(len=4)::readheader(MSnvar+7),scname

  ! ****************************************************************
  !  "IO" is an integer, unit number of a file open for read
  !  "ser" is the identifier of the series to be read (ser should
  !  be zero at input)
  ! ****************************************************************

  ! Validation

  if(ser.ne.0) then ! if the series already exists
     MSmessage='error in READ: the identifier (1) is not the &
          &identifier of a null series'
     MSintmess=0
     MSintmess(1)=ser
     call ERROR
  endif

  ! Read the label and the number of terms
  read(IO,999) file
  read(IO,999) file
  read(IO,999) file
  read(IO,999) file
  read(IO,999) file,nterm
999 format(A29,I6)

  read(IO,999) file
  read(IO,999) file

  if(nterm.eq.0) return

  ! Read the name of the variables in the file and check them against the
  !  current ones
```

```
  read(IO,59) (readheader(i),i=1,MSnvar+7)
59 format(1x,37A4)

  ! if the header does no correspond
  do i=1,MSnvar+7
     if(readheader(i).ne.MSheader(i)) then
        MSmessage= 'error in READ: the variable (1) is identified&
             & by (a) and not by (b)'
        MSintmess=0
        MSintmess(1)=i
        MScharmess(1)=readheader(i)
        MScharmess(2)=MSheader(i)
        call ERROR

     endif
  enddo

  ! Read the terms and store them

  do i=1,nterm
     read(IO,MSformread) scname,(arg(k),k=1,MSnvt),(exp(k),k=1,MSnvp),coef

     sc=0
     if(scname.eq.'sin(') sc=1
     call STORE(ser,sc,arg,exp,coef)
  enddo

end subroutine READ

!************************************************************************!
! subroutine ERASE(ser)                                                  !
! The subroutine ERASE destroys the series A and reclaims the room       !
! previously occupied by it                                              !
!                                                                        !
!************************************************************************!
subroutine ERASE(ser)
  use MSTABLES
  implicit none
  integer,intent(inout)::ser
  integer endser,k

  ! Validation
  if(ser.eq.0) return ! if the series is empty -> nothing to do
  ser=abs(ser)
  if(ser.gt.MSnmaxser) then ! if there are more series than permitted
     MSmessage='error in ERASE: the identifier (1) is&
          & not valid '
     MSintmess=0
     MSintmess(1)=ser
     call ERROR
  endif
```

```
  ! Reclaim the room

  endser=MSbegin(ser)+MSlength(ser)
  if(endser.ne.MSfree) then
     call MOVEBLOCK(MSfree-endser,endser,MSbegin(ser))
  endif

  ! Update the table of beginning of series

  do k=1,MSnser
     if(MSbegin(k).gt.MSbegin(ser)) MSbegin(k)=MSbegin(k)-MSlength(ser)
  enddo
  MSfree=MSfree-MSlength(ser)
  MSbegin(ser)=0
  MSlength(ser)=0
  ser=0

end subroutine ERASE


!*********************************************************************!
! subroutine COPY(ser1,ser2)                                          !
! The subroutine COPY takes a copy of the series ser1 and identifies  !
! it by ser2. The identifier of ser2 must be 0 (empty series) before  !
! the call                                                            !
!                                                                     !
!*********************************************************************!
subroutine COPY(ser1,ser2)
  use MSTABLES
  implicit none
  integer,intent(in):: ser1
  integer,intent(inout):: ser2
  integer absser

  ! Validation
  if(ser1.eq.0) return ! If the series ser1 is empty
  absser=abs(ser1)
  if(absser.gt.MSnmaxser) then ! if there are more series than permitted
     MSmessage='error in COPY: the identifier (1) is&
          & not valid '
     MSintmess=0
     MSintmess(1)=ser1
     call ERROR
  endif

  if(ser2.ne.0) then ! if the series ser2 already exists
     MSmessage='error in COPY: the identifier (1) is&
          & not zero '
     MSintmess=0
     MSintmess(1)=ser2
     call ERROR
```

```
  endif



  ! copy the series at the end of the stack

  call INIT(ser2)
  MSlength(ser2)=MSlength(absser)
  call MOVEBLOCK(MSlength(absser),MSbegin(absser),MSfree)
  MSfree=MSfree+MSlength(absser)
  if(ser1.lt.0) ser2=-ser2

end subroutine COPY


!*********************************************************************!
! subroutine RENAME(ser1,ser2)                                        !
! The subroutine RENAME exchanges the identifiers of the series ser1  !
! and ser2                                                            !
!                                                                     !
!*********************************************************************!
subroutine RENAMEE(ser1,ser2)
  implicit none
  integer,intent(inout)::ser1,ser2
  integer:: temp

  temp=ser1
  ser1=ser2
  ser2=temp

end subroutine RENAMEE


!*********************************************************************!
! subroutine CUTEPS(ser,epsilon)                                      !
! The subroutine CUTEPS drops, in the series identified by ser, all   !
! the terms with a coefficient less (in absolute value) than epsilon  !
!                                                                     !
!*********************************************************************!
subroutine CUTEPS(ser,epsilon)
  use MSTABLES
  implicit none
  double precision,intent(in)::epsilon
  integer,intent(inout):: ser
  integer j,k,knew,count,endser,absser

  ! Validation
  if(ser.eq.0) return ! if the series is empty, nothing to do
  absser=abs(ser)
  if(absser.gt.MSnmaxser) then ! if there are more series than permitted
     MSmessage='error in CUTEPS: the identifier (1) is&
          & not valid '
```

```
     MSintmess=0
     MSintmess(1)=ser
     call ERROR
  endif

  ! scan the series: the retained terms are copied at the beginning
  !  of the series

  endser=MSbegin(absser)+MSlength(absser)
  count=0
  do k=MSbegin(absser),endser-1
     if(abs(MStabcoef(k)).ge.epsilon) then
        knew=k-count
        do j=1,MSnwords
           MSTABLE(j,knew)=MSTABLE(j,k)
        enddo
        MStabcoef(knew)=MStabcoef(k)
     else
        count=count+1
     endif
  enddo

  ! Reclame the room if needed

  if(count.eq.0) return

  if(endser.ne.MSfree) then
     call MOVEBLOCK(MSfree-endser,endser,endser-count)
     do k=1,MSnser
        if(MSbegin(k).gt.MSbegin(absser)) MSbegin(k)=MSbegin(k)-count
     enddo
  endif

  if(count.eq.MSlength(absser)) then
     MSbegin(absser)=0
     ser=0
  endif

  MSlength(absser)=MSlength(absser)-count
  MSfree=MSfree-count

end subroutine CUTEPS


!*********************************************************************!
! subroutine CHANGE_NAME(oldname,newname)                             !
! The subroutine CHANGE_NAME substitute to the name oldname of a      !
! variable the new name newname                                       !
!                                                                     !
!*********************************************************************!
subroutine CHANGE_NAME(oldname,newname)
  use MSTABLES
```

```
  implicit none
  integer :: k
  character(len=4),intent(in):: oldname,newname

  ! Validation

  do k=1,MSnvt
     if(oldname.eq.MSnamevt(k)) goto 100
  enddo

  do k=1,MSnvp
     if(oldname.eq.MSnamevp(k)) goto 200
  enddo

  MSmessage='error in CHANGE_NAME: the argument (a)&
       &is not the name of a variable '
  MSintmess=0
  MScharmess(1)=oldname
  call ERROR

100 MSnamevt(k)=newname
  return

200 MSnamevp(k)=newname

end subroutine CHANGE_NAME


!*********************************************************************!
! function INDEXTRIG(name)                                            !
! The integer function INDEXTRIG gives the rank of the trigonometric  !
! variable the name of which is name                                  !
!                                                                     !
!*********************************************************************!
integer function INDEXTRIG(name)
  use MSTABLES
  implicit none
  integer :: k
  character(len=4),intent(in):: name

  ! Validation

  do k=1,MSnvt
     if(name.eq.MSnamevt(k)) goto 100
  enddo

  MSmessage='error in INDEXTRIG: the argument (a) &
       &is not the name of a trig variable '
  MSintmess=0
  MScharmess(1)=name
  call ERROR
```

```
100 INDEXTRIG=k

end function INDEXTRIG


!*********************************************************************!
! function INDEXPOL(name)                                             !
! The integer function INDEXPOL gives the rank of the polynomial      !
! variable the name of which is name                                  !
!                                                                     !
!*********************************************************************!
integer function INDEXPOL(name)
  use MSTABLES
  implicit none
  integer :: k
  character(len=4),intent(in):: name

  ! Validation

  do k=1,MSnvp
     if(name.eq.MSnamevp(k)) goto 100
  enddo

  MSmessage='error in INDEXPOL: the argument (a) &
       &is not the name of a pol variable '
  MSintmess=0
  MScharmess(1)=name
  call ERROR

100 INDEXPOL=k

end function INDEXPOL


!*********************************************************************!
! subroutine SCALE(ser,alpha)                                         !
! The subroutine SCALE multiplies the series indentified by ser by    !
! the factor alpha                                                    !
!                                                                     !
!*********************************************************************!
subroutine SCALE(ser,alpha)
  use MSTABLES
  implicit none
  double precision,intent(in)::alpha
  integer,intent(in):: ser
  integer j,k,absser

  ! Validation

  if(ser.eq.0) return ! if the series is empty, nothing to do
  absser=abs(ser)
  if(absser.gt.MSnmaxser) then ! if there are more series than permitted
```

```
     MSmessage='error in SCALE: the identifier (1) is&
          & not valid '
     MSintmess(1)=ser
     call ERROR
  endif

  ! Multiplication by alpha

  do k=MSbegin(absser),MSbegin(absser)+MSlength(absser)-1
     MStabcoef(k)=alpha*MStabcoef(k)
  enddo

  ! drop the terms which are smaller than epsilon

  call CUTEPS(ser,MSaccuracy)

end subroutine SCALE


!*********************************************************************!
! subroutine SCALEP(ser,namepol,alpha)                                !
! The subroutine SCALEP scales, in the series identified by ser, the  !
! polynomial variable identified by namepol by the factor alpha       !
!                                                                     !
!*********************************************************************!
subroutine SCALEP(ser,namepol,alpha)
  use MSTABLES
  implicit none
  double precision,intent(in)::alpha
  integer,intent(in):: ser
  integer j,k,indexpol,sc,arg(MSnvt),exp(MSnvp),absser
  character(len=4),intent(in):: namepol

  ! Validation

  if(ser.eq.0) return ! if the series is empty, nothing to do
  absser=abs(ser)
  if(absser.gt.MSnmaxser) then ! if there are more series than permitted
     MSmessage='error in SCALE: the identifier (1) is&
          & not valid '
     MSintmess=0
     MSintmess(1)=ser
     call ERROR
  endif

  ! Find the index of the polynomial variable

  do indexpol=1,MSnvp
     if(namepol.eq.MSnamevp(indexpol)) goto 10
  enddo
  MSmessage='error in SCALEP: the argument (a) is not the name &
       &of a polynomial variable'
```

```
  MSintmess=0
  MScharmess(1)=namepol
  call ERROR

  ! Multiplication by alpha**argp(indexpol)

10 do k=MSbegin(absser),MSbegin(absser)+MSlength(absser)-1
     call DECODE(MSTABLE(1,k),sc,arg,exp)
     MStabcoef(k)=MStabcoef(k)*alpha**exp(indexpol)
  enddo

  ! drop the terms which are smaller than epsilon

  call CUTEPS(ser,MSaccuracy)

  ! reorder the series if needed

  if(ser.lt.0) call ORDER_SIZE(ser)

end subroutine SCALEP


!*********************************************************************!
! subroutine EVALP(ser,namepol,alpha)                                 !
! The subroutine EVALP gives, in the series indentified by ser, the   !
! value alpha of the polynomial variable idetified by namepol         !
!                                                                     !
!*********************************************************************!
subroutine EVALP(ser,namepol,alpha)
  use MSTABLES
  implicit none
  double precision,intent(in)::alpha
  integer,intent(in):: ser
  integer:: sc,arg(MSnvt),exp(MSnvp),temp=0,NBTERM
  integer::j,k,absser,indexpol
  double precision:: coef
  character(len=4),intent(in):: namepol

  ! Validation
  if(ser.eq.0) return ! if the series is empty, nothing to do
  absser=abs(ser)
  if(absser.gt.MSnmaxser) then ! if there are more series than permitted
     MSmessage='error in SCALE: the identifier (1) is&
          & not valid '
     MSintmess=0
     MSintmess(1)=ser
     call ERROR
  endif

  ! Find the index of the polynomial variable

  do indexpol=1,MSnvp
```

```
     if(namepol.eq.MSnamevp(indexpol)) goto 10
  enddo
  MSmessage='error in SCALEP: the argument is not the name &
       &of a polynomial variable'
  MSintmess=0
  call ERROR

  ! Multiplication by alpha**exp(indexpol) and setting exp(indexpol)
  !  to zero

10 do k=1,NBTERM(ser)
     call FETCH(ser,k,sc,arg,exp,coef)
     coef=coef*alpha**exp(indexpol)
     exp(indexpol)=0
     call STORE(temp,sc,arg,exp,coef)
  enddo
  call ERASE(ser)
  call RENAMEE(ser,temp)


  ! drop the terms which are smaller than epsilon

  call CUTEPS(ser,MSaccuracy)

  ! reorder the series if needed

  if(ser.lt.0) call ORDER_SIZE(ser)

end subroutine EVALP


!*********************************************************************!
! subroutine EVALT(ser,nametrig,alpha)                                !
! The subroutine EVALT gives, in the series indentified by ser, the   !
! value alpha of the trigonometric variable identified by nametrig    !
!                                                                     !
!*********************************************************************!
SUBROUTINE EVALT(ser,nametrig,alpha)
  use MSTABLES
  implicit none
  integer,intent(inout):: ser
  character(len=4),intent(in)::nametrig
  double precision,intent(in):: alpha
  integer:: CoCo=0,CoSi=0,SiCo=0,SiSi=0,Res=0
  integer:: sc,arg(MSnvt),exp(MSnvp),NBTERM
  integer:: i,imax,j,INDEXTRIG,indT
  double precision:: coef

  indT=INDEXTRIG(nametrig)

  ! ********************************************************************
  !  Remind some trigonometric rules
```

```
!   cos(X+m.sig) = cos(X)*cos(m.sig) - sin(X)*sin(m.sig)
!  sin(X+m.sig) = sin(X)*cos(m.sig) + cos(X)*sin(m.sig)
!
! ********************************************************************

! get the maximum multiplier (in absolute value) of "nametrig"
!  and store in the result terms independant of it

imax=0
do j=1,NBTERM(ser)
   call FETCH(ser,j,sc,arg,exp,coef)
   if(abs(arg(indT)).gt.imax) imax=abs(arg(indT))
   if(arg(indT).eq.0) call STORE(Res,sc,arg,exp,coef)
enddo

! Find the terms in cos(i.trig +X)
!  and form CoCo = cos(X) and CoSi = sin(X)
!
!  NOTE: We avoid constructing one term of CoCo and then one term of
!  CoSi and then a new term of CoCo etc.

do i=1,imax
   do j=1,NBTERM(ser)
      call FETCH(ser,j,sc,arg,exp,coef)
      if(abs(arg(indT)).eq.i.and.sc.eq.0) then
         arg(indT)=0
         call STORE(CoCo,0,arg,exp,coef)
      endif
   enddo

   do j=1,NBTERM(ser)
      call FETCH(ser,j,sc,arg,exp,coef)
      if(abs(arg(indT)).eq.i.and.sc.eq.0) then
         coef=coef*sign(1.d0,dble(arg(indT)))
         arg(indT)=0
         call STORE(CoSi,1,arg,exp,coef)
      endif
   enddo

   ! Products

   call ACUM(CoCo,Res,cos(dble(i)*alpha))
   call ACUM(CoSi,Res,-sin(dble(i)*alpha))
   call ERASE(CoCo)
   call ERASE(CoSi)

   ! Find the terms in sin(i.trig +X)
   !  and form SiCo = cos(X) and SiSi = sin(X)

   do j=1,NBTERM(ser)
      call FETCH(ser,j,sc,arg,exp,coef)
      if(abs(arg(indT)).eq.i.and.sc.eq.1) then
```

```
            coef=coef*sign(1.d0,dble(arg(indT)))
            arg(indT)=0
            call STORE(SiCo,0,arg,exp,coef)
         endif
      enddo

      do j=1,NBTERM(ser)
         call FETCH(ser,j,sc,arg,exp,coef)
         if(abs(arg(indT)).eq.i.and.sc.eq.1) then
            arg(indT)=0
            call STORE(SiSi,1,arg,exp,coef)
         endif
      enddo

      ! Products

      call ACUM(SiSi,Res,cos(dble(i)*alpha))
      call ACUM(SiCo,Res,sin(dble(i)*alpha))
      call ERASE(SiCo)
      call ERASE(SiSi)

   enddo

   call ERASE(ser)
   call RENAMEE(ser,Res)
   call CUTEPS(ser,MSaccuracy)

end subroutine EVALT


!*********************************************************************!
! subroutine XMON(ser,namepol,exponent)                               !
! The subroutine XMON multiplies the series identified by ser by the  !
! monomial variable identified by namepol to the power exponent       !
!                                                                     !
!*********************************************************************!
subroutine XMON(ser,namepol,exponent)
   use MSTABLES
   implicit none
   double precision::coef
   integer,intent(in):: ser,exponent
   integer k,temp,sc,arg(MSnvt),exp(MSnvp),NBTERM,indexpol,absser
   character(len=4), intent(in):: namepol

   ! Validation
   if(ser.eq.0) return ! if the series is empty, nothing to do
   absser=abs(ser)
   if(absser.gt.MSnmaxser) then ! if there are more series than permitted
      MSmessage='error in XMON: the identifier (1) is&
           & not valid '
      MSintmess=0
      MSintmess(1)=ser
```

```
     call ERROR
  endif

  ! Find the index of the polynomial variable

  do indexpol=1,MSnvp
     if(namepol.eq.MSnamevp(indexpol)) goto 10
  enddo
  MSmessage='error in XMON: the argument char(1) is not the name of&
       & a polynomial variable '
  MSintmess=0
  MScharmess(1)=namepol
  call ERROR


  ! Multiplication  by polynomial variable

10 temp=0

  do k=1,NBTERM(absser)
     call FETCH(ser,k,sc,arg,exp,coef)
     exp(indexpol)=exp(indexpol)+exponent
     call STORE(temp,sc,arg,exp,coef)
  enddo

  call ERASE(ser)
  call RENAMEE(ser,temp)

end subroutine XMON


!*********************************************************************!
! subroutine ACUM(ser,sum,alpha)                                      !
! The subroutine ACUM replaces the series sum by its previous content  !
! augmented by the product alpha*ser                                  !
!                                                                     !
!*********************************************************************!
subroutine ACUM(ser,sum,alpha)
  use MSTABLES
  implicit none
  double precision,intent(in)::alpha
  integer,intent(in):: ser,sum
  integer k,sc,arg(MSnvt),exp(MSnvp),j,absser

  ! Validation
  if(abs(sum).gt.MSnmaxser) then ! if there are more series than permitted
     MSmessage='error in ACUM: the identifier (1) of the &
          &series sum is not valid '
     MSintmess=0
     MSintmess(1)=sum
     call ERROR
  endif
```

```
  if(ser.eq.0) return ! if the series is empty, nothing to do

  absser=abs(ser)
  if(absser.gt.MSnmaxser) then ! if there are more series than permitted
     MSmessage='error in ACUM: the identifier (1) of the &
          &series to be added is not valid '
     MSintmess=0
     MSintmess(1)=ser
     call ERROR
  endif

  ! sum should be ordered by code

  call ORDER_CODE(sum)
  call MOVE_TO_END(sum)

  ! sum and ser should not be the same

  if(absser.eq.sum) then
     MSmessage='error in ACUM: the identifier (1) and (2) of &
          & the two series are equal'
     MSintmess=0
     MSintmess(1)=ser
     MSintmess(2)=sum
     call ERROR
  endif

  ! insert the terms of ser into sum

  do k=MSbegin(absser),MSbegin(absser)+MSlength(absser)-1
     call PUTTRM(sum,MSTABLE(1,k),alpha*MStabcoef(k))
  enddo

  call CUTEPS(sum,MSaccuracy)

end subroutine ACUM


!*********************************************************************!
! subroutine PROD(serA,serB,product,alpha)                            !
! The subroutine PROD replaces the series product by its previous     !
! content augmented by the product alpha*serA*serB                    !
!                                                                     !
!*********************************************************************!
subroutine PROD(serA,serB,product,alpha)
  use MSTABLES
  implicit none
  double precision,intent(in)::alpha
  double precision::coefA,coefB,coefR
  integer,intent(in):: serA,serB,product
  integer k,kA,scA,argA(MSnvt),expA(MSnvp)
```

```
integer kB,scB,argB(MSnvt),expB(MSnvp)
integer expR(MSnvp),argS(MSnvt),argD(MSnvt)
integer absserA,absserB
integer NBTERM

! Nothing to do if A or B are null
if(serA.eq.0.or.serB.eq.0) return

! Validation
absserA=abs(serA)
if(absserA.gt.MSnmaxser) then ! if there are more series than permitted
   MSmessage='error in PROD: the identifier (1) of the &
        &first series is not valid '
   MSintmess=0
   MSintmess(1)=serA
   call ERROR
endif

absserB=abs(serB)
if(absserB.gt.MSnmaxser) then ! if there are more series than permitted
   MSmessage='error in PROD: the identifier (1) of the &
        &second series is not valid '
   MSintmess=0
   MSintmess(1)=serB
   call ERROR
endif

if(abs(product).gt.MSnmaxser) then ! if there are more series than permitted
   MSmessage='error in PROD: the identifier (1) of the &
        & series product is not valid '
   MSintmess=0
   MSintmess(1)=product
   call ERROR
endif


! product should be ordered according to the code

call ORDER_CODE(product)

! do loops on the terms of A and B
do kA=1,NBTERM(absserA)
   call FETCH(absserA,kA,scA,argA,expA,coefA)
   do kB=1,NBTERM(absserB)
      call FETCH(absserB,kB,scB,argB,expB,coefB )

      ! Sum of the exponents
      expR=expA+expB
      ! Product of the coefficients
      coefR=coefA*coefB*alpha*0.5d0

      ! Sum and dif of the arguments
```

```
        argS=argA+argB
        argD=argA-argB


        ! The result is in sine if scA.ne.scB
        if(scA.ne.scB) then
           call STORE(product,1,argS,expR,coefR)
           if(scA.ne.1) coefR=-coefR
           call STORE(product,1,argD,expR,coefR)
        else
           ! in cosine otherwise
           call STORE(product,0,argD,expR,coefR)
           if(scA.eq.1) coefR=-coefR
           call STORE(product,0,argS,expR,coefR)
        endif
     enddo
  enddo

  call CUTEPS(product,MSaccuracy)

end subroutine PROD


!*********************************************************************!
! subroutine PRODCT(serA,serB,product,alpha,eps)                      !
! The subroutine PRODCT acts as the subroutine PROD, except that the  !
! product of individual terms is discarded each time the absolute     !
! value of its coefficient is smaller than eps                        !
!                                                                     !
!*********************************************************************!
subroutine PRODCT(serA,serB,product,alpha,eps)
  use MSTABLES
  implicit none
  double precision,intent(in)::alpha,eps
  double precision::coefA,coefB,coefR
  integer,intent(inout):: serA,serB,product
  integer k,kA,scA,argA(MSnvt),expA(MSnvp)
  integer kB,scB,argB(MSnvt),expB(MSnvp)
  integer expR(MSnvp),argS(MSnvt),argD(MSnvt)
  integer absserA,absserB
  integer NBTERM
  logical:: done

  ! Nothing to do if A or B are null
  if(serA.eq.0.or.serB.eq.0) return

  ! Validation
  absserA=abs(serA)
  if(absserA.gt.MSnmaxser) then ! if there are more series than permitted
     MSmessage='error in PROD: the identifier (1) of the &
          &first series is not valid '
     MSintmess=0
```

```
   MSintmess(1)=serA
   call ERROR
endif

absserB=abs(serB)
if(absserB.gt.MSnmaxser) then ! if there are more series than permitted
   MSmessage='error in PROD: the identifier (1) of the &
        &second series is not valid '
   MSintmess=0
   MSintmess(1)=serB
   call ERROR
endif

! factor should be ordered according to size

if(serA.gt.0) call ORDER_SIZE(serA)
if(serB.gt.0) call ORDER_SIZE(serB)

! Validation

if(abs(product).gt.MSnmaxser) then ! if there are more series than permitted
   MSmessage='error in PROD: the identifier (1) of the &
        & series product is not valid '
   MSintmess=0
   MSintmess(1)=product
   call ERROR
endif

! product should be ordered according to the code

if(product.lt.0) call ORDER_CODE(product)


done=.false.

! do loops on the terms of A and B

do kA=1,NBTERM(absserA)
   call FETCH(absserA,kA,scA,argA,expA,coefA)
   do kB=1,NBTERM(absserB)

      ! Product of the coefficients, end the loop on the terms of B if the
      !  product is too small

      call FETCH(absserB,kB,scB,argB,expB,coefB )
      coefR=coefA*coefB*alpha*0.5d0

      if(abs(coefR).lt.eps.and.kB.eq.1) done=.true.
      if(abs(coefR).lt.eps) exit

      ! Sum of the exponents
      expR=expA+expB
```

```
        ! Sum and dif of the arguments
        argS=argA+argB
        argD=argA-argB


        ! The result is in sine if scA.ne.scB
        if(scA.ne.scB) then
           call STORE(product,1,argS,expR,coefR)
           if(scA.ne.1) coefR=-coefR
           call STORE(product,1,argD,expR,coefR)
        else
           ! in cosine otherwise
           call STORE(product,0,argD,expR,coefR)
           if(scA.eq.1) coefR=-coefR
           call STORE(product,0,argS,expR,coefR)
        endif
     enddo
     if(done) exit
  enddo

  call CUTEPS(product,MSaccuracy)

end subroutine PRODCT


!*********************************************************************!
! subroutine PDERT(ser,sumder,nametrig)                               !
! The subroutine PDERT adds to the series sumder the derivative of ser !
! with respect to the trigonometric variable identified by nametrig    !
!                                                                     !
!*********************************************************************!
subroutine PDERT(ser,sumder,nametrig)
  use MSTABLES
  implicit none
  double precision::coef
  integer,intent(in):: ser,sumder
  integer k,sc,arg(MSnvt),exp(MSnvp),sign,NBTERM,index,absser
  integer INDEXTRIG
  character(len=4),intent(in)::nametrig

  ! Validation
  if(sumder.ne.0) then ! if the series sumder already exists
     MSmessage='error in PDERT: the identifier (1) of the &
          &sumder is not zero '
     MSintmess=0
     MSintmess(1)=sumder
     call ERROR
  endif
```

```
  if(ser.eq.0) return ! if the series is empty, nothing to do
  absser=abs(ser)
  if(absser.gt.MSnmaxser) then ! if there are more series than permitted
     MSmessage='error in PDERT: the identifier (1) of the &
          &series to be differentiated is not valid '
     MSintmess=0
     MSintmess(1)=ser
     call ERROR
  endif


  ! Find the index of the variable

  index=INDEXTRIG(nametrig)

  ! Differentiate

10 do k=1,NBTERM(absser)
     call FETCH(ser,k,sc,arg,exp,coef)
     if(arg(index).ne.0) then
        sign=1.d0
        if(sc.eq.0) sign=-sign
        coef=sign*coef*dble(arg(index))
        sc=1-sc
        call STORE(sumder,sc,arg,exp,coef)
     endif
  enddo

end subroutine PDERT


!*********************************************************************!
! subroutine PDERP(ser,sumder,namepol)                                !
! The subroutine PDERP adds to the series sumder the derivative of ser !
! with respect to the polynomial variable identified by namepol       !
!                                                                     !
!*********************************************************************!
subroutine PDERP(ser,sumder,namepol)
  use MSTABLES
  implicit none
  double precision::coef
  integer,intent(in):: ser,sumder
  integer k,sc,arg(MSnvt),exp(MSnvp),sign,NBTERM,index,absser
  integer INDEXPOL
  character(len=4),intent(in)::namepol


  ! Validation
  if(sumder.ne.0) then ! if the series sumder already exists
     MSmessage='error in PDERP: the identifier (1) of the &
          &sumder is not zero '
     MSintmess=0
```

```
     MSintmess(1)=sumder
     call ERROR
  endif

  if(ser.eq.0) return ! if the series is empty, nothing to do
  absser=abs(ser)
  if(absser.gt.MSnmaxser) then ! if there are more series than permitted
     MSmessage='error in PDERP: the identifier (1) of the &
          &series to be differentiated is not valid '
     MSintmess=0
     MSintmess(1)=ser
     call ERROR
  endif
  ! Find the index of the polynomial variable

  index=INDEXPOL(namepol)

  ! Differentiate

10 do k=1,NBTERM(absser)
     call FETCH(ser,k,sc,arg,exp,coef)
     if(exp(index).ne.0) then
        coef=coef*dble(exp(index))
        exp(index)=exp(index)-1
        call STORE(sumder,sc,arg,exp,coef)
     endif
  enddo

end subroutine PDERP


!*********************************************************************!
! subroutine ORDER_SIZE(ser)                                          !
! The subroutine ORDER_SIZE reorders the series ser according to the  !
! size of the absolute value of the coefficients                      !
! The identificator of the series is set to a negative value          !
!                                                                     !
!*********************************************************************!
subroutine ORDER_SIZE(ser)
  ! ******************************************************************
  !
  !  The sorting algorithm is the "diminishing increment sort"
  !  (see Knuth, the art of computer programming, vol 3, p 85
  !  the increments are h_1, h_{s+1} = 1+3h_s, in decreasing
  !  order
  !  the first one h_t is such that h_{t+2} .ge. NBTERM(A)
  !
  ! ******************************************************************
  use MSTABLES
  implicit none
  integer, intent(inout):: ser
  double precision:: coef,sizej,sizei
```

```
integer:: inc,inclimit,beg,nt,termj,termi,arg(MSnwords)
integer::j,k,NBTERM

! If the series is empty, do nothing

if(ser.eq.0) return

! validation

if(ser.gt.MSnmaxser) then
   MSmessage='error in ORDER_SIZE: the identifier of the series&
        & is (1) and is not valid '
   MSintmess(1)=ser
   call ERROR
endif

nt=NBTERM(abs(ser))
beg=MSbegin(abs(ser))

! mark the series as ordered by size

if(ser.gt.0) ser = -ser
if(nt.eq.1) return

! inc is the length of the first increment

inc=1
inclimit=13
do
   if(inclimit.gt.nt) exit
   inc=3*inc+1
   inclimit=3*inclimit+1
enddo


! exit if the step with inc=1 has been performed
do
   if(inc.le.0) exit

   ! step D2 of Knuth: scan the terms of the series

   do j=inc+1,nt
      termj=beg+j-1
      termi=beg+j-inc-1
      sizej=abs(MStabcoef(termj))
      ! step D3 of Knuth: save termj
      do k=1,MSnwords
         arg(k)=MSTABLE(k,termj)
      enddo
      coef=MStabcoef(termj)
      ! step D3 of Knuth: compare termj and termi
```

```
100     sizei=abs(MStabcoef(termi))
        if(sizei.lt.sizej) then
           ! step D5 of Knuth: if needed, move up termi and decrease i

           do k=1,MSnwords
              MSTABLE(k,termi+inc)=MSTABLE(k,termi)
           enddo
           MStabcoef(termi+inc)=MStabcoef(termi)
           termi=termi-inc
           if(termi.ge.beg) goto 100
        endif

        ! step D6 of Knuth: insert termj
        do k=1,MSnwords
           MSTABLE(k,termi+inc)=arg(k)
        enddo
        MStabcoef(termi+inc)=coef

     enddo

     ! loop on the diminishing increment

     inc=(inc-1)/3
  enddo

end subroutine ORDER_SIZE


!*********************************************************************!
! subroutine ORDER_CODE(ser)                                          !
! The subroutine ORDER_CODE reorders the series ser according to the  !
! lexographic (alphabetic) order of the keys (according to the code)  !
! The identificator of the series is set to a positive value          !
!                                                                     !
!*********************************************************************!
subroutine ORDER_CODE(ser)
  ! ********************************************************************
  !
  !  The sorting algorithm is the "diminishing increment sort"
  !  (see Knuth, the art of computer programming, vol 3, p 85
  !  the increments are h_1, h_{s+1} = 1+3h_s, in decreasing
  !  order
  !  the first one h_t is such that h_{t+2} .ge. NBTERM(A)
  !
  ! ********************************************************************
  use MSTABLES
  implicit none
  integer, intent(inout):: ser
  double precision:: coef
  integer:: codej(MSnwords),codei(MSnwords)
  integer:: inc,inclimit,beg,nt,termj,termi,arg(MSnwords)
  integer::j,k,n,NBTERM
```

```
  ! If the series is empty, do nothing

  if(ser.eq.0) return

  ! mark the series as ordered by code
  if(ser.lt.0) ser=-ser

  ! validation

  if(ser.gt.MSnmaxser) then
     MSmessage='error in ORDER_CODE: the identifier of the series&
          & is (1) and is not valid '
     MSintmess(1)=ser
     call ERROR
  endif

  ! inc is the length of the first increment

  nt=NBTERM(ser)
  beg=MSbegin(ser)
  inc=1
  inclimit=13
  do
     if(inclimit.gt.nt) exit
     inc=3*inc+1
     inclimit=3*inclimit+1
  enddo

  ! exit if the step with inc=1 has been performed
  do
     if(inc.le.0) exit

     ! step D2 of Knuth: scan the terms of the series

     do j=inc+1,nt
        termj=beg+j-1
        termi=beg+j-inc-1

        ! step D3 of Knuth: save termj
        do k=1,MSnwords
           codej(k)=MSTABLE(k,termj)
           arg(k)=MSTABLE(k,termj)
        enddo
        coef=MStabcoef(termj)
        ! step D3 of Knuth: compare termj and termi

100     do k=1,MSnwords
           codei(k)=MSTABLE(k,termi)
           if(codei(k).lt.codej(k)) goto 200
           if(codei(k).gt.codej(k)) then
              ! step D5 of Knuth: if needed, move up termi and decrease i
```

```
              do n=1,MSnwords
                 MSTABLE(n,termi+inc)=MSTABLE(n,termi)
              enddo
              MStabcoef(termi+inc)=MStabcoef(termi)
              termi=termi-inc
              exit
           endif
        enddo
        if(termi.ge.beg) goto 100

        ! step D6 of Knuth: insert termj
200     do n=1,MSnwords
           MSTABLE(n,termi+inc)=arg(n)
        enddo
        MStabcoef(termi+inc)=coef
     enddo

     ! loop on the diminishing increment

     inc=(inc-1)/3
  enddo
end subroutine ORDER_CODE


!*********************************************************************!
! subroutine SUBSTITUTE(ser,insert,name)                              !
! The subroutine SUBSTITUTE replaces in the series ser the variable   !
! name by the series insert. The variable name should not appear to a !
! negative power in the series ser                                    !
!                                                                     !
!*********************************************************************!
subroutine SUBSTITUTE(ser,insert,name)
  use MSTABLES
  implicit none
  integer,intent(in):: insert
  integer,intent(inout):: ser
  integer::j,k,max,indexp,sc,arg(MSnvt),exp(MSnvp),NBTERM,INDEXPOL
  integer::power=0,res=0,temp=0
  character(len=4),intent(in):: name
  double precision:: coef


  ! validation

  if(ser.eq.0) return ! if the series is empty, nothing to do

  if(abs(ser).gt.MSnmaxser) then ! if there are more series than permitted
     MSmessage='error in SUBSTITUTE: the identifier of the series&
          & in which the substitution should be done is (1) &
          &and is not valid '
     MSintmess(1)=ser
```

```
   call ERROR
endif

if(ser.lt.0) call ORDER_CODE(ser)

if(abs(insert).gt.MSnmaxser) then
   MSmessage='error in SUBSTITUTE: the identifier of the series&
        & to be inserted is (1) and is not valid '
   MSintmess(1)=insert
   call ERROR
endif

! Find the index of the polynomial variable

indexp=INDEXPOL(name)

! Find the maximum power to be computed

max=0
do k=1,NBTERM(ser)
   call FETCH(ser,k,sc,arg,exp,coef)
   if(exp(indexp).lt.0) then
      MSmessage='error in SUBSTITUTE: there is a term such that&
           & the power (1) of the variable (a) is negative'
      MScharmess(1)=name
      MSintmess=exp(indexp)
      call ERROR
   endif
   if(exp(indexp).gt.max) max=exp(indexp)
   if(exp(indexp).eq.0) call STORE(res,sc,arg,exp,coef)
enddo

! "insert" to the power k

call CONSTANT(power,1.d0)
do k=1,max
   call PROD(power,insert,temp,1.d0)
   call ERASE(power)
   call RENAMEE(power,temp)

   ! get the terms with exp = k

   do j=1,NBTERM(ser)
      call FETCH(ser,j,sc,arg,exp,coef)
      if(exp(indexp).eq.k) &
           &call STORE(temp,sc,arg,exp,coef)
   enddo

   call ACUM(temp,ser,-1.d0)
   call XMON(temp,name,-k)
   call PROD(temp,power,res,1.d0)
   call ERASE(temp)
```

```
  enddo

  call ERASE(power)
  call ERASE(ser)
  call RENAMEE(ser,res)

end subroutine SUBSTITUTE


!*********************************************************************!
! subroutine V_ERASE(a,order)                                         !
! The subroutine V_ERASE is the same as ERASE for vectorial operations !
!                                                                     !
!*********************************************************************!
subroutine V_ERASE(a,order)
  use MSTABLES
  implicit none
  integer,intent(in) :: order
  integer,intent(inout):: a(0:order)
  integer n

  ! validation of "order"
  if(order.lt.0.or.order.gt.MSordmax) then
     MSmessage='error in V_ERASE: the order (1) is out of bound'
     MSintmess(1)=order
     call ERROR
  endif

  ! erase

  do n=0,order
     call ERASE(a(n))
  enddo
end subroutine V_ERASE


!*********************************************************************!
! subroutine V_RENAMEE(a,b,order)                                     !
! The subroutine V_RENAMME is the same as RENAME for vectorial        !
! operations                                                          !
!                                                                     !
!*********************************************************************!
subroutine V_RENAMEE(a,b,order)
  use MSTABLES
  implicit none
  integer,intent(in) :: order
  integer,intent(inout):: a(0:order),b(0:order)

  integer n

  ! validation of "order"
```

```
  if(order.lt.0.or.order.gt.MSordmax) then
     MSmessage='error in V_RENAMEE: the order (1) is out of bound'
     MSintmess(1)=order
     call ERROR
  endif

  ! RENAMEE

  do n=0,order
     call RENAMEE(a(n),b(n))
  enddo
end subroutine V_RENAMEE


!*********************************************************************!
! subroutine V_CUTEPS(a,epsilon,order)                                !
! The subroutine V_CUTEPS is the same as CUTEPS for vectorial         !
! operations                                                          !
!                                                                     !
!*********************************************************************!
subroutine V_CUTEPS(a,epsilon,order)
  use MSTABLES
  implicit none
  integer,intent(in) :: order
  integer,intent(inout):: a(0:order)
  double precision,intent(in):: epsilon
  integer n

  ! validation of "order"
  if(order.lt.0.or.order.gt.MSordmax) then
     MSmessage='error in V_CUTEPS: the order (1) is out of bound'
     MSintmess(1)=order
     call ERROR
  endif

  ! cuteps

  do n=0,order
     call CUTEPS(a(n),epsilon)
  enddo
end subroutine V_CUTEPS


!*********************************************************************!
! subroutine V_COPY(a,b,order)                                        !
! The subroutine V_COPY is the same as COPY for vectorial operations  !
!                                                                     !
!*********************************************************************!
subroutine V_COPY(a,b,order)
  use MSTABLES
  implicit none
  integer,intent(in) :: order,a(0:order)
```

```
  integer,intent(inout):: b(0:order)
  integer n

  ! validation of "order"
  if(order.lt.0.or.order.gt.MSordmax) then
     MSmessage='error in V_COPY: the order (1) is out of bound'
     MSintmess(1)=order
     call ERROR
  endif

  ! copy

  do n=0,order
     call COPY(a(n),b(n))
  enddo
end subroutine V_COPY


!*********************************************************************!
! subroutine V_SCALE(a,alpha,order)                                   !
! The subroutine V_SCALE is the same as SCALE for vectorial operations !
!                                                                     !
!*********************************************************************!
subroutine V_SCALE(a,alpha,order)
  use MSTABLES
  implicit none
  integer,intent(in) :: order
  integer,intent(inout):: a(0:order)
  integer n
  double precision,intent(in) :: alpha

  ! validation of "order"
  if(order.lt.0.or.order.gt.MSordmax) then
     MSmessage='error in V_SCALE: the order (1) is out of bound'
     MSintmess(1)=order
     call ERROR
  endif

  ! scale

  do n=0,order
     call SCALE(a(n),alpha)
  enddo
end subroutine V_SCALE


!*********************************************************************!
! subroutine V_PROD(a,b,c,coef,order)                                 !
! The subroutine V_PROD is the same as PROD for vectorial operations  !
!                                                                     !
!*********************************************************************!
subroutine V_PROD(a,b,c,coef,order)
```

```
  use MSTABLES
  implicit none
  integer,intent(in) :: order
  integer,intent(in) :: a(0:order),b(0:order)
  integer,intent(inout):: c(0:order)
  double precision ,intent(in)::coef
  integer n,j

  ! validation of "order"
  if(order.lt.0.or.order.gt.MSordmax) then
     MSmessage='error in V_PROD: the order (1) is out of bound'
     MSintmess(1)=order
     call ERROR
  endif

  ! product

  do n=0,order
     do j=0,n
        call PROD(a(j),b(n-j),c(n),coef)
     enddo
  enddo
end subroutine V_PROD


!*********************************************************************!
! subroutine V_ACUM(a,b,coef,order)                                   !
! The subroutine V_ACUM is the same as ACUM for vectorial operations  !
!                                                                     !
!*********************************************************************!
subroutine V_ACUM(a,b,coef,order)
  use MSTABLES
  implicit none
  integer,intent(in) :: order
  integer,intent(in) :: a(0:order)
  integer,intent(inout):: b(0:order)
  double precision ,intent(in)::coef
  integer n

  ! validation of "order"
  if(order.lt.0.or.order.gt.MSordmax) then
     MSmessage='error in V_ACUM: the order (1) is out of bound'
     MSintmess(1)=order
     call ERROR
  endif

  ! sum

  do n=0,order
     call ACUM(a(n),b(n),coef)
  enddo
end subroutine V_ACUM
```

```
!*********************************************************************!
! subroutine V_PRINT(iounit,ser,label,order)                          !
! The subroutine V_PRINT is the same as PRINT for vectorial operations !
!                                                                     !
!*********************************************************************!
subroutine V_PRINT(iounit,ser,label,order)
  use MSTABLES
  implicit none
  integer,intent(in) :: iounit,order
  integer,intent(in):: ser(0:order)
  integer n
  character(len=*),intent(in)::label

  ! validation of "order"
  if(order.lt.0.or.order.gt.MSordmax) then
     MSmessage='error in V_PRINT: the order (1) is out of bound'
     MSintmess(1)=order
     call ERROR
  endif

  ! print

  do n=0,order
     call PRINT(iounit,ser(n),label,n)
  enddo
end subroutine V_PRINT


!*********************************************************************!
! subroutine V_SUBSTITUTE(ser,insert,name,order)                      !
! The subroutine V_SUBSTITUTE replaces in the series ser(n) the       !
! variable name by the expansion up to order "order" of the series    !
! insert=sum(insert(k)).                                              !
!                                                                     !
!*********************************************************************!
subroutine V_SUBSTITUTE (ser,insert,name,order)
  ! ******************************************************************
  !
  !  in the series "ser", substitute the series "insert"
  !  to the polynomial variable "name"
  !
  !  both "ser" and "insert" are vectorial series of order
  !  "order"
  !
  ! ******************************************************************
  use MSTABLES
  implicit none
  integer,intent(in):: insert(0:MSordmax),order
  integer,intent(inout):: ser(0:MSordmax)
  character(len=4),intent(in)::name
```

```
integer:: sc,arg(MSnvt),exp(MSnvp),NBTERM
double precision:: coef
integer::power(0:MSordmax)=0, RES(0:MSordmax)=0
integer:: temp=0,temp2(0:MSordmax)=0
integer:: k,n,j,powermax,INDEXPOL,indexp

! Validation

if(order.lt.0.or.order.gt.MSordmax) then ! validation of order
   MSmessage='error on V_substitute: the order (1) is out&
        & of bound'
   MSintmess=0
   MSintmess(1)=order
   call ERROR
endif

do k=0,order
   if(abs(ser(k)).gt.MSnmaxser) then ! if there are more series than permitted
      MSmessage='error in V_substitute: the identifier of ser(index (1)) &
           & is (2) and is not valid '
      MSintmess=0
      MSintmess(1)=k
      MSintmess(2)=ser(k)
      call ERROR
   endif
   if(ser(k).lt.0) call ORDER_CODE(ser(k))
enddo

do k=0,order
   if(abs(insert(k)).gt.MSnmaxser) then ! if there are more series than permitted
      MSmessage='error in  V_substitute: the identifier of insert(index (1))&
           & is (2) and is not valid '
      MSintmess=0
      MSintmess(1)=k
      MSintmess(2)=insert(k)
      call ERROR
   endif
   if(insert(k).lt.0) call ORDER_CODE(insert(k))
enddo

! Index of the variable to be substituted

indexp=INDEXPOL(name)

! loop on the orders of ser

do n=0,order

   !  At each order compute the highest power of the variable
   !  to be substituted
   !  Store in the result (RES(n)) the terms independant of this variable
```

```
   powermax=0
   do j=1,NBTERM(ser(n))

      ! check that there is no negative powers of the variable

      call FETCH(ser(n),j,sc,arg,exp,coef)
      if(exp(indexp).lt.0) then
         MSmessage='error in  V_substitute: negative power (1)of the&
              & variable (a) to be substituted'
         MSintmess=0
         MSintmess=exp(indexp)
         MScharmess=name
         call ERROR
      endif

      if(exp(indexp).eq.0) call STORE(RES(n),sc,arg,exp,coef)
      if(exp(indexp).gt.powermax) powermax=exp(indexp)
   enddo


   !  At the order n we have to compute the powers of "insert" up to order
   !  "order - n


   call CONSTANT(power(0),1.d0)
   do k=1,powermax

      ! powers of "insert"

      call V_PROD(power,insert,temp2,1.d0,order-n)
      call V_ERASE(power,order-n)
      call V_RENAMEE(power,temp2,order-n)

      ! selection of the terms with power "k" of the variable

      do j=1,NBTERM(ser(n))
         call FETCH(ser(n),j,sc,arg,exp,coef)
         if(exp(indexp).eq.k) &
              &call STORE(temp,sc,arg,exp,coef)
      enddo

      call ACUM(temp,ser(n),-1.d0)
      call XMON(temp,name,-k)
      do j=0,order-n
         call PROD(power(j),temp,RES(n+j),1.d0)
      enddo
      call ERASE(temp)

   enddo
   call V_ERASE(power,order)
enddo
```

```
  call V_ERASE(ser,order)
  call V_RENAMEE(ser,RES,order)

end subroutine V_SUBSTITUTE


!*********************************************************************!
! subroutine POWER(ser,exponent,order)                                !
!                                                                     !
!*********************************************************************!
subroutine POWER(ser,exponent,order)
  ! ******************************************************************
  !
  !  Replace "ser" by "ser" to the power "exponent"
  !
  !  "ser" is a vector containing the successive orders
  !  of a function expanded in powers of a "small parameter".
  !  The result is computed up to order "order" in this parameter.
  !
  !  ser(0) must be a pure number
  !
  ! ******************************************************************
  use MSTABLES
  implicit none
  double precision, intent(in):: exponent
  double precision:: coef,scaling
  integer, intent(in)::order
  integer, intent(inout):: ser(0:order)
  integer:: temp(0:MSordmax)=0
  integer:: n,k,NBTERM,sc,arg(MSnvt),exp(MSnvp)

  ! Validation
  ! Check if ser(0) is the unit series

  if(NBTERM(ser(0)).ne.1) then ! if there are more than one term in ser(0)
     MSmessage='error in POWER: the number of terms of the zero&
          & order of the input array is (1)'
     MSintmess=NBTERM(ser(0))
     call ERROR
  else
     call FETCH(ser(0),1,sc,arg,exp,scaling)

     do n=1,MSnvt
        if(arg(n).ne.0) then ! if the arguments of ser(0) aren't zero
           MSmessage='error in POWER: the trig arg number (1)&
                & of the zero order of the input array is (2)'
           MSintmess(1)=n
           MSintmess(2)=arg(n)
           call ERROR
        endif
     enddo
     do n=1,MSnvp
```

```
        if(exp(n).ne.0) then ! if the exponents of ser(0) aren't zero
           MSmessage='error in POWER: the pol arg number (1)&
                & of the zero order of the input array is (2)'
           MSintmess(1)=n
           MSintmess(2)=exp(n)
           call ERROR
        endif

     enddo
  endif

  call V_SCALE(ser,1.d0/scaling,order)

  ! start the computation at order 0

  call ACUM(ser(0),temp(0),1.d0)
  if(n.eq.0) return

  ! do loop on the order

  do n=1,order
     coef=exponent*dble(n)
     call ACUM(ser(n),temp(n),coef)
     if(n.gt.1) then
        do k=1,n-1
           coef=exponent*dble(n-k)
           call PROD(temp(k),ser(n-k),temp(n),coef)
           coef=-coef/exponent
           call PROD(ser(k),temp(n-k),temp(n),coef)
        enddo
     endif
     coef=1.d0/dble(n)
     call SCALE(temp(n),coef)
  enddo

  call V_ERASE(ser,order)
  call V_RENAMEE(ser,temp,order)

  scaling=scaling**exponent
  call V_SCALE(ser,scaling,order)


end subroutine POWER


!*********************************************************************!
! subroutine POLAR_TO_CART(ser,nameamp,nametrig,nameX,nameY)          !
! The subroutine POLAR_TO_CART transforms in the series ser, the polar !
! coordinates nameamp and nametrig into cartesian coordinates nameX    !
! and nameY                                                           !
!                                                                     !
!*********************************************************************!
```

```
SUBROUTINE POLAR_TO_CART(ser,nameamp,nametrig,nameX,nameY)
  use MSTABLES
  implicit none
  character(len=4),intent(in)::nametrig,nameamp,nameX,nameY
  integer,intent(inout):: ser
  integer:: CO=0,SI=0,COi=0,SIi=0,Res=0,sum=0,temp=0
  integer:: CoCo=0,CoSi=0,SiCo=0,SiSi=0,ONE=0,tempC=0,tempS=0
  integer:: sc,arg(MSnvt),exp(MSnvp),NBTERM
  integer:: k,n,i,imax,j,INDEXTRIG,INDEXPOL,indT,indX,indY,indA
  double precision:: coef,sign

  ! Validation

  if(abs(ser).gt.MSnmaxser) then ! if there are more series than permitted
     MSmessage='error in SPOLAR_TO_CART: the identifier of the series&
          & in which the substitution should be done is (1) &
          &and is not valid '
     MSintmess(1)=ser
     call ERROR
  endif

  indT=INDEXTRIG(nametrig)
  indA=INDEXPOL(nameamp)
  indX=INDEXPOL(nameX)
  indY=INDEXPOL(nameY)


  ! ****************************************************************
  !
  !  Replace R^(ir+2n) cos.or.sin(ir+phi)
  !  by (X^2+Y^2)^n.R^ir.cos.or.sin(ir+phi)
  !
  ! ****************************************************************

  ! form sum = X^2+Y^2

  call CONSTANT(sum,1.d0);call XMON(sum,nameX,2)
  call CONSTANT(temp,1.d0);call XMON(temp,nameY,2)
  call ACUM(temp,sum,1.d0);call ERASE(temp)

  ! Subtract from the exponent of R the absolute value of the trig argument
  !  of r in order to get "2n" and divide by 2 to get "n"

  do k=1,NBTERM(ser)
     call FETCH(ser,k,sc,arg,exp,coef)
     exp(indA)=exp(indA)-abs(arg(indT))
     if(exp(indA).lt.0.or.exp(indA).ne.2*(exp(indA)/2)) then
        MSmessage='error in POLAR_TO_CART: the exponent &
             &of (a) is (1) and the argument of (b) is (2)'
        MSintmess(1)=exp(indA)+abs(arg(indT))
        MSintmess(2)=arg(indT)
        MScharmess(1)=nameamp
```

```
      MScharmess(2)=nametrig
      call ERROR
   else
      exp(indA)=exp(indA)/2
      call STORE(temp,sc,arg,exp,coef)
   endif
enddo
call ERASE(ser)
call RENAMEE(ser,temp)

! substitute sum to "a^n"

call SUBSTITUTE(ser,sum,nameamp)
call ERASE(sum)
call CUTEPS(ser,MSaccuracy)

! ********************************************************************
!
!   R^m*cos(X+m.r) = cos(X)*R^m*cos(m.r) - sin(X)*R^m*sin(m.r)
!  R^m*sin(X+m.r) = sin(X)*R^m*cos(m.r) + cos(X)*R^m*sin(m.r)
!
!
! ********************************************************************

! form the series CO = cos r and SI = sin r

call CONSTANT(ONE,1.d0)
call COPY(ONE,CO)
call XMON(CO,nameX,1)
call COPY(ONE,SI)
call XMON(SI,nameY,1)

! get the maximum multiplier of "nametrig"
!  and store in the result terms independant of it

imax=0
do j=1,NBTERM(ser)
   call FETCH(ser,j,sc,arg,exp,coef)
   if(abs(arg(indT)).gt.imax) imax=abs(arg(indT))
   if(arg(indT).eq.0) call STORE(Res,sc,arg,exp,coef)
enddo


! initialize COi = cos(0 r) and SIi = sin(0 r)

call COPY(ONE,COi)
call ERASE(ONE)

do i=1,imax

   ! form the new COi = cos(i r) and SIi = sin(i r)
```

```
call PROD(COi,CO,tempC,1.d0)
call PROD(SIi,SI,tempC,-1.d0)
call PROD(COi,SI,tempS,1.d0)
call PROD(SIi,CO,tempS,1.d0)
call ERASE(COI)
call ERASE(SIi)
call RENAMEE(COi,tempC)
call RENAMEE(SIi,tempS)

! Find the terms in cos(i.r +X)
!  and form CoCo = cos(X) and CoSi = [sign(i)].sin(X)
!
!  NOTE: We avoid constructing one term of CoCo and then one term of
!  CoSi and then a new term of CoCo etc.

do j=1,NBTERM(ser)
   call FETCH(ser,j,sc,arg,exp,coef)
   if(abs(arg(indT)).eq.i.and.sc.eq.0) then
      arg(indT)=0
      call STORE(CoCo,0,arg,exp,coef)
   endif
enddo


do j=1,NBTERM(ser)
   call FETCH(ser,j,sc,arg,exp,coef)
   if(abs(arg(indT)).eq.i.and.sc.eq.0) then
      sign=dble(arg(indT))/dble(abs(arg(indT)))
      arg(indT)=0
      call STORE(CoSi,1,arg,exp,coef*sign)
   endif
enddo

! Find the terms in sin(i.r +X)
!  and form SiCo = [sign(i)].cos(X) and SiSi = sin(X)

do j=1,NBTERM(ser)
   call FETCH(ser,j,sc,arg,exp,coef)
   if(abs(arg(indT)).eq.i.and.sc.eq.1) then
      sign=dble(arg(indT))/dble(abs(arg(indT)))
      arg(indT)=0
      call STORE(SiCo,0,arg,exp,coef*sign)
   endif
enddo

do j=1,NBTERM(ser)
   call FETCH(ser,j,sc,arg,exp,coef)
   if(abs(arg(indT)).eq.i.and.sc.eq.1) then
      arg(indT)=0
      call STORE(SiSi,1,arg,exp,coef)
   endif
```

```
     enddo

     ! Products

     call PROD(CoCo,COi,Res,1.d0)
     call PROD(CoSi,SIi,Res,-1.d0)
     call PROD(SiSi,COi,Res,1.d0)
     call PROD(SiCo,SIi,Res,1.d0)

     call ERASE(CoCo)
     call ERASE(CoSi)
     call ERASE(SiCo)
     call ERASE(SiSi)

  enddo

  call ERASE(CO)
  call ERASE(SI)
  call ERASE(COi)
  call ERASE(SIi)

  call ERASE(ser)
  call RENAMEE(ser,Res)
  call CUTEPS(ser,MSaccuracy)

end subroutine POLAR_TO_CART


!*********************************************************************!
! subroutine CART_TO_POLAR(ser,nameamp,nametrig,nameX,nameY)          !
! The subroutine CART_TO_POLAR transforms in the series ser, the      !
! cartesian coordinates nameX and nameY into polar coordinates nameamp !
! and nametrig                                                        !
!                                                                     !
!*********************************************************************!
SUBROUTINE CART_TO_POLAR(ser,nameamp,nametrig,nameX,nameY)
  use MSTABLES
  implicit none
  character(len=4),intent(in)::nametrig,nameX,nameY,nameamp
  integer,intent(inout):: ser
  integer:: arg(MSnvt),exp(MSnvp)
  integer:: X=0,Y=0
  integer:: INDEXTRIG,INDEXPOL,indT,indA

  ! Validation

  if(abs(ser).gt.MSnmaxser) then ! if there are more series than permitted
     MSmessage='error in SUBSTITUTE: the identifier of the series&
          & in which the substitution should be done is (1) &
          &and is not valid '
     MSintmess(1)=ser
     call ERROR
```

```
  endif

  indT=INDEXTRIG(nametrig)
  indA=INDEXPOL(nameamp)

  ! form the X =R*cos(r) and Y = R*sin(r)

  arg=0 ; exp=0
  arg(indT)=1
  exp(indA)=1
  call STORE(X,0,arg,exp,1.d0)
  call STORE(Y,1,arg,exp,1.d0)

  ! substitute X for nameX and Y for  nameY

  call SUBSTITUTE(ser,X,nameX)
  call SUBSTITUTE(ser,Y,nameY)
  call ERASE(X)
  call ERASE(Y)


  call CUTEPS(ser,MSaccuracy)

end subroutine CART_TO_POLAR


!*****************************************************************************!
! subroutine dev2B_fr(ser,order)                                              !
!                                                                             !
!*****************************************************************************!
subroutine dev2B_fr(ser,order)
  use MSTABLES
  implicit none
  integer,intent(in):: ser(0:MSordmax),order
  integer::WF(MSordmax)=0,WR=0,sc,arg(MSnvt)=0,exp(MSnvp)=0
  integer:: sinf=0,sin2f=0,der=0,res=0,temp=0
  integer:: k,i,j,mm,jj,inde,indr,indf
  integer:: BINOM,NBTERM
  double precision fact,coef


  ! validation

  if(order.lt.0.or.order.gt.MSordmax) then
     MSmessage='error on dev2B_fr: the order (1) is out&
          & of bound'
     MSintmess=0
     MSintmess(1)=order
     call ERROR
  endif

  do k=0,order
```

```
     if(ser(k).lt.0.or.ser(k).gt.MSnmaxser) then
        MSmessage='error in dev2B_fr: the identifier of ser(see (1))&
             & is (2) and is not valid '
        MSintmess=0
        MSintmess(1)=k
        MSintmess(2)=ser(k)
        call ERROR
     endif
  enddo

  ! ************************************************************************
  !
  !  Find the indexes of  "e" the eccentricity,
  !  "r" the ratio r/a,
  !  "f" the true anomaly
  !
  ! ************************************************************************

  do inde=1,MSnvp
     if(MSnamevp(inde).eq.'   e') goto 100
  enddo

  MSmessage='error on dev2B_fr: no polynomial variable is named e'
  MSintmess=0
  call ERROR

100 do indr=1,MSnvp
     if(MSnamevp(indr).eq.'   r') goto 200
  enddo

  MSmessage='error on dev2B_fr: no polynomial variable is named r'
  MSintmess=0
  call ERROR

200 do indf=1,MSnvt
     if(MSnamevt(indf).eq.'   f') goto 300
  enddo

  MSmessage='error on dev2B_fr: no trigonometric variable is named f'
  MSintmess=0
  call ERROR

  ! ************************************************************************
  !
  !  If order = 0, return after setting the value of 'r' to 1
  !
  ! ************************************************************************

300 if(order.eq.0) then
     do k=1,NBTERM(ser)
        call FETCH(ser,k,sc,arg,exp,coef)
        exp(indr)=0
```

```
      call STORE(temp,sc,arg,exp,coef)
   enddo
   call ERASE(ser)
   call RENAMEE(ser,temp)
   return
endif

! ************************************************************************
!
!  Formation of the generator of the Lie transform which defines
!  the expansion
!
!  dr/de = -cos f  =  RW/e
!
!  df/de = (4sin f +e sin 2f)/2(1-e^2) = FW/e
!
!
! ************************************************************************

arg=0
exp=0
arg(indf)=1
exp(inde)=1
call STORE(WR,0,arg,exp,-1.d0)

exp(inde)=0
call STORE(sinf,1,arg,exp,2.d0)
arg(indf)=2
call STORE(sin2f,1,arg,exp,0.5d0)

fact=1.d0
do k=1,order
   if((k-1).gt.0) fact=fact*dble(k-1)
   if(k.ne.2*(k/2)) then
      call ACUM(sinf,WF(k),fact)
   else
      call ACUM(sin2f,WF(k),fact)
   endif

   call XMON(WF(k),'   e',k)
enddo

call ERASE(sinf)
call ERASE(sin2f)

! ************************************************************************
!
!  Expansion of "ser" by Lie transform
!
! ************************************************************************

! scaling the input series
```

```
fact=1.d0
do k=1,order
   fact=fact*dble(k)
   call SCALE(ser(k),fact)
enddo

! contribution of the derivatives w.r.t "r"

do k=1,order
   do i=k,order
      mm=order+k-i
      call ACUM(ser(mm),res,1.d0)
      call PDERP(ser(mm-1),der,'   r')
      call PROD(der,WR,res,1.d0)
      call ERASE(der)

      ! contribution of the derivatives w.r.t. "f"

      jj=mm-k+1
      do j=1,jj
         fact=dble(BINOM(mm-k,j-1))
         call PDERT(ser(mm-j),der,'   f')
         call PROD(der,WF(j),res,fact)
         call ERASE(der)
      enddo
      call ERASE(ser(mm))
      call RENAMEE(res,ser(mm))
   enddo
   call ERASE(WF(order-k+1))
enddo
call ERASE(WR)

! ************************************************************************
!
!  Renormalisation: set to zero the exponent of r
!   divide by the factorial
!
! ************************************************************************

fact=1.d0
do k=0,order
   do i=1,NBTERM(ser(k))
      call FETCH(ser(k),i,sc,arg,exp,coef)
      exp(indr)=0
      call STORE(res,sc,arg,exp,coef*fact)
   enddo
   call ERASE(ser(k))
   call RENAMEE(ser(k),res)
   call CUTEPS(ser(k),MSaccuracy)
   fact=fact/dble(k+1)
enddo
```

```
end subroutine dev2B_fr


!*********************************************************************!
! function BINOM(n,m)                                                 !
! The integer function BINOM calculates the binomial coefficient      !
! n!/m!(n-m)!                                                         !
!*********************************************************************!
integer function BINOM(n,m)
  use MSTABLES
  implicit none
  integer,intent(in):: n,m
  integer:: j

  ! validation

  if(n.lt.0.or.m.lt.0.or.m.gt.n) then
     MSmessage= 'error in BINOM: the input (1), (2) is not&
          & valid'
     MSintmess=0
     MSintmess(1)=n
     MSintmess(2)=m
     call ERROR
  endif

  BINOM=1
  if(m.eq.0.or.m.eq.n) return

  if(m.ge.(n-m)) then
     do j=1,n-m
        BINOM=BINOM*(m+j)/j
     enddo
  else
     do j=1,m
        BINOM=BINOM*(n-m+j)/j
     enddo
  endif

end function BINOM


!*********************************************************************!
! subroutine DEV_ANGLE(ser,angle,pert,order)                          !
!                                                                     !
!*********************************************************************!
subroutine DEV_ANGLE(ser,angle,pert,order)
  ! ********************************************************************
  !
  !  The subroutine substitute in the series 'input' the angular variable
  !  the name of which is 'angle' by the expression angle + pert
  !  (where pert is a series assumed to be small).
```

```
!
!  The substitution is done by expansion is series up to the power 'order'
!  of 'pert'.
!
!  The result is accumulated in 'output'
!
! ********************************************************************
use MSTABLES
implicit none
integer , intent(in):: pert,order
integer , intent(inout):: ser
character(len=4),intent(in)::angle
integer:: tempser=0, power(MSordmax)=0,classCS=0,classSC=0
integer:: sc,arg(MSnvt),exp(MSnvp),NBTERM,temparg(MSnvt),tempsc
integer:: k,j,indangle,INDEXTRIG
double precision:: coef,mult,sign,fact


! ********************************************************************
!
!  Validation
!
! ********************************************************************

if(ser.eq.0) return

if(abs(ser).gt.MSnmaxser) then
   MSmessage='error in DEV_ANGLE: the identifier (1) of the input&
        & series is not valid '
   MSintmess=0
   MSintmess(1)=ser
   call ERROR
endif

if(abs(pert).gt.MSnmaxser) then
   MSmessage='error in DEV_ANGLE: the identifier (1) of the pert&
        & series is not valid '
   MSintmess=0
   MSintmess(1)=pert
   call ERROR
endif


if(order.le.0.or.order.gt.MSordmax) then
   MSmessage= 'error in DEV_ANGLE : order (see (1)) is not &
        &between 1 and MSordmax (see (2))'
   MSintmess=0
   MSintmess(1)=order
   MSintmess(2)=MSordmax
   call ERROR
endif
```

```
  ! ************************************************************************
  !
  !  form the powers of 'pert' up to the power 'order'
  !
  ! ************************************************************************



  call COPY(pert,power(1))

  if(order.ge.2) then
     do k=2,order
        call PROD(power(k-1),pert,power(k),1.d0)
     enddo
  endif

  call COPY(ser,tempser)
  call ERASE(ser)
  indangle=INDEXTRIG(angle)

  ! ************************************************************************
  !
  !  Select a term containing the angular variable 'angle'
  !
  ! ************************************************************************


10 do k=1,NBTERM(tempser)
     call FETCH(tempser,k,sc,arg,exp,coef)
     if(arg(indangle).ne.0) then
        mult=dfloat(arg(indangle))
        tempsc=sc
        do j=1,MSnvt
           temparg(j)=arg(j)
        enddo
        goto 100  ! a term containing 'angle' is found
     endif
  enddo

  ! ************************************************************************
  !
  !  No terms containing 'angle' are left;
  !  add the terms not containing 'angle', clean up and exit
  !
  ! ************************************************************************

  call ACUM(tempser,ser, 1.d0)
  call ERASE(tempser)
  call V_ERASE(power,order)
  return
```

```
  ! ********************************************************************
  !
  !  Select the class of terms with the same trigonometric function
  !
  ! ********************************************************************


100 do k=1,NBTERM(tempser)
     call FETCH(tempser,k,sc,arg,exp,coef)
     if(sc.ne.tempsc) goto 200
     do j=1,MSnvt
        if(temparg(j).ne.arg(j)) goto 200
     enddo

     call STORE(classCS,sc,arg,exp,coef)

200  CONTINUE
  enddo


  ! ********************************************************************
  !
  !  cos(j*angle + j*pert + phi) = cos(j*angle + phi)*cos(j*pert)
  !  - sin(j*angle + phi)*sin(j*pert)
  !
  ! ********************************************************************

  IF(tempsc.eq.0) THEN

     !  form the series 'class2' with a sine in place of the cosine


     call PDERT(classCS,classSC,angle)
     call SCALE(classSC,-1.d0/mult)
     call ACUM(classCS,ser,1.d0)

     fact=1.d0
     do k=1,order
        fact=fact/dfloat(k)
        if(k.eq.2*(k/2)) then

           !  form cos(j*angle + phi)*cos(j*pert) up to power 'order'
           sign=fact
           if(k.ne.4*(k/4)) sign=-sign
           call PROD(classCS,power(k),ser,sign*mult**k)
        else

           !  form -sin(j*angle + phi)*cos(j*pert) up to power 'order'
           sign=fact
           if((k-1).ne.4*(k/4)) sign=-sign
           call PROD(classSC,power(k),ser,-sign*mult**k)
```

```
      endif
    enddo


    ! ************************************************************************
    !
    !  sin(j*angle + j*pert + phi) = sin(j*angle + phi)*cos(j*pert)
    !  + cos(j*angle + phi)*sin(j*pert)
    !
    ! ************************************************************************

  ELSE

    !  form the series 'class2' with a cosine in place of the sine


    call PDERT(classCS,classSC,angle)
    call SCALE(classSC, 1.d0/mult)
    call ACUM(classCS,ser,1.d0)

    fact=1.d0
    do k=1,order
      fact=fact/dfloat(k)
      if(k.eq.2*(k/2)) then

        !  form sin(j*angle + phi)*cos(j*pert) up to power 'order'
        sign=fact
        if(k.ne.4*(k/4)) sign=-fact
        call PROD(classCS,power(k),ser,sign*mult**k)
      else

        !  form cos(j*angle + phi)*sin(j*pert) up to power 'order'
        sign=fact
        if((k-1).ne.4*(k/4)) sign=-fact
        call PROD(classSC,power(k),ser, sign*mult**k)
      endif
    enddo

  ENDIF

  call ACUM(classCS,tempser,-1.d0)
  call ERASE(classCS)
  call ERASE(classSC)


  GOTO 10 ! find another term containing 'angle'

end subroutine DEV_ANGLE
```

Annexe C

Données des séries

C.1 υ-And, système moyennisé

Ci-dessous se trouvent les données concernant la série (5.4) vue au Chapitre 5.

sc	p_1	p_2	E_1	E_2	Coefficient
0	0	0	0	0	0.1028814873602426D+01
0	0	0	2	0	0.5028538710029945D-01
0	0	0	0	2	0.5028538710029944D-01
0	1	-1	1	1	-0.4076228931886004D-01
0	1	-1	1	3	-0.1222586149960201D+00
0	0	0	2	2	0.1197001849250330D+00
0	0	0	0	4	0.6709635675735098D-01
0	1	-1	3	1	-0.4221317158954681D-01
0	2	-2	2	2	0.2213605213725824D-01
0	0	0	4	0	-0.7246264294834475D-02
0	1	-1	1	5	-0.2524696341343997D+00
0	0	0	2	4	0.2467628264483285D+00
0	1	-1	3	3	-0.1921063144213280D+00
0	2	-2	2	4	0.9925686521957974D-01
0	0	0	0	6	0.8758635391968309D-01
0	2	-2	4	2	0.1933648622251537D-01
0	0	0	4	2	0.1536832637434396D-01
0	3	-3	3	3	-0.1169269078562161D-01
0	1	-1	5	1	0.6836391071059079D-02
0	0	0	6	0	-0.1935582947009399D-02
0	1	-1	3	5	-0.6196370195770786D+00
0	0	0	2	6	0.5074002052071770D+00
0	1	-1	1	7	-0.4655700431956326D+00
0	2	-2	2	6	0.2885649356365574D+00
0	0	0	4	4	0.1494291348471382D+00
0	2	-2	4	4	0.1398141027469650D+00

suite de la page précédente

sc	p_1	p_2	E_1	E_2	Coefficient
0	0	0	0	8	0.1197831351071404D+00
0	3	-3	3	5	-0.7214092929111357D-01
0	1	-1	5	3	-0.2874229576993508D-01
0	3	-3	5	3	-0.1010146673582717D-01
0	0	0	6	2	-0.8840759409828003D-02
0	4	-4	4	4	0.6329373494027248D-02
0	1	-1	7	1	0.3460048337093939D-02
0	2	-2	6	2	-0.2898632538440724D-02
0	0	0	8	0	-0.9809864482075091D-04
0	1	-1	3	7	-0.1736049482521746D+01
0	0	0	2	8	0.1044430546083579D+01
0	1	-1	1	9	-0.8333781073867842D+00
0	2	-2	2	8	0.7075428074981931D+00
0	0	0	4	6	0.6419395602614903D+00
0	2	-2	4	6	0.6291827345254666D+00
0	1	-1	5	5	-0.3522102144470720D+00
0	3	-3	3	7	-0.2760798557690555D+00
0	0	0	0	10	0.1730054022073606D+00
0	3	-3	5	5	-0.1014306076159650D+00
0	4	-4	4	6	0.5065986173740239D-01
0	2	-2	6	4	0.2702481583358223D-01
0	1	-1	7	3	0.2003014807787981D-01
0	0	0	6	4	0.1047652322537889D-01
0	4	-4	6	4	0.5810306687876316D-02
0	0	0	8	2	-0.3637127163866981D-02
0	5	-5	5	5	-0.3547073606586902D-02
0	2	-2	8	2	-0.2626033743514865D-02
0	3	-3	7	3	0.1141528383858028D-02
0	1	-1	9	1	0.2702463941348789D-03
0	0	0	10	0	0.4164862158317965D-04
0	1	-1	3	9	-0.4494853277789366D+01
0	2	-2	4	8	0.2264937330206745D+01
0	0	0	4	8	0.2185560143597253D+01
0	0	0	2	10	0.2141733558394088D+01
0	1	-1	5	7	-0.1896033810861642D+01
0	2	-2	2	10	0.1601280940539568D+01
0	1	-1	1	11	-0.1489035024909751D+01
0	3	-3	3	9	-0.8548457895032535D+00
0	3	-3	5	7	-0.5861272857481159D+00
0	2	-2	6	6	0.3699469098621153D+00
0	0	0	6	6	0.2855172475317053D+00
0	0	0	0	12	0.2613228083735695D+00
0	4	-4	4	8	0.2429470703230471D+00
0	4	-4	6	6	0.7413306257577218D-01
0	5	-5	5	7	-0.3523828350986871D-01
0	3	-3	7	5	-0.2471265545969524D-01
0	0	0	8	4	-0.2221759372473406D-01
0	2	-2	8	4	-0.1848856965626547D-01
0	1	-1	7	5	-0.1373717214640005D-01
0	1	-1	9	3	0.9596353503517214D-02

suite de la page précédente

sc	p_1	p_2	E_1	E_2	Coefficient
0	5	-5	7	5	-0.3553483704778352D-02
0	6	-6	6	6	0.2053892837676242D-02
0	3	-3	9	3	0.1900111782325308D-02
0	4	-4	8	4	-0.3610752537537270D-03
0	2	-2	10	2	-0.3320784753799520D-03
0	0	0	10	2	-0.2462069543517898D-03
0	1	-1	11	1	-0.1324939863629992D-03
0	0	0	12	0	0.1053152556202829D-04

TABLE C.1 – Les données concernant υ-And

C.2 Séries utilisées pour les tests

Ci-dessous se trouvent les données concernant les séries que nous avons utilisées pour tester les sous-routines vectorielles.

Il s'agit de deux séries, que nous noterons respectivement A et B issues de celle sus-présentée. Elles ont été obtenues en choisissant de placer un terme sur deux dans A et dans B.
La série A (respectivement B) contient donc le $1^{er}, 3^e, \ldots$ (respectivement le $2^e, 4^e, \ldots$) terme de la série de départ.
De plus, nous avons tronqué les séries à l'ordre 8 pour des raisons évidentes d'affichage.

sc	p_1	p_2	E_1	E_2	Coefficient
0	0	0	0	0	0.1028814873602426D+01
0	0	0	0	2	0.5028538710029944D-01
0	1	-1	1	3	-0.1222586149960201D+00
0	0	0	0	4	0.6709635675735098D-01
0	2	-2	2	2	0.2213605213725824D-01
0	1	-1	1	5	-0.2524696341343997D+00
0	1	-1	3	3	-0.1921063144213280D+00
0	0	0	0	6	0.8758635391968309D-01
0	0	0	4	2	0.1536832637434396D-01
0	1	-1	5	1	0.6836391071059079D-02
0	1	-1	3	5	-0.6196370195770786D+00
0	1	-1	1	7	-0.4655700431956326D+00
0	0	0	4	4	0.1494291348471382D+00
0	0	0	0	8	0.1197831351071404D+00
0	1	-1	5	3	-0.2874229576993508D-01
0	0	0	6	2	-0.8840759409828003D-02
0	1	-1	7	1	0.3460048337093939D-02
0	0	0	8	0	-0.9809864482075091D-04

TABLE C.2 – La série A

sc	p_1	p_2	E_1	E_2	Coefficient
0	0	0	2	0	0.5028538710029945D-01
0	1	-1	1	1	-0.4076228931886004D-01
0	0	0	2	2	0.1197001849250330D+00
0	1	-1	3	1	-0.4221317158954681D-01
0	0	0	4	0	-0.7246264294834475D-02
0	0	0	2	4	0.2467628264483285D+00
0	2	-2	2	4	0.9925686521957974D-01
0	2	-2	4	2	0.1933648622251537D-01
0	3	-3	3	3	-0.1169269078562161D-01
0	0	0	6	0	-0.1935582947009399D-02
0	0	0	2	6	0.5074002052071770D+00
0	2	-2	2	6	0.2885649356365574D+00
0	2	-2	4	4	0.1398141027469650D+00
0	3	-3	3	5	-0.7214092929111357D-01
0	3	-3	5	3	-0.1010146673582717D-01
0	4	-4	4	4	0.6329373494027248D-02
0	2	-2	6	2	-0.2898632538440724D-02

TABLE C.3 – La série B

Annexe D

Test V_POISSON

Comme nous l'avons vu au Chapitre 4, nous avons créé deux sous-routines permettant de calculer les parenthèses de Poisson entre deux fonctions. La première version, classique, a été validée à l'aide d'un exemple simple. La seconde version, vectorielle, a été validée comme suit :
Prenons les séries A et B décrites à l'Annexe C. Nous avons effectué le calcul des crochets de Poisson $\{A, B\}$ avec la sous-routine `POISSON` (supposée correcte grâce aux tests précédents) ainsi qu'avec `V_POISSON`. Le lecteur peut constater ci-dessous que les résultats concordent, moyennant quelques différentes d'un ordre inférieur à la précision machine dans les coefficients.

Voici les résultats de la version classique :

```
      SERIES     Poisson brackets         1
       NUMBER OF TERMS :    151


      p1  p2       E1  E2             COEF
sin(   1  -1 ) (   1   2)   -0.5215977519644021D-10
sin(   1  -1 ) (   1   4)   -0.1391947476901391D-09
sin(   1  -1 ) (   1   6)   -0.2725534073118022D-09
sin(   1  -1 ) (   1   8)   -0.4969922086550716D-09
sin(   1  -1 ) (   2   3)   -0.1794043464677527D-09
sin(   1  -1 ) (   2   5)   -0.6954622817661499D-09
sin(   1  -1 ) (   2   7)   -0.1823677094096726D-08
sin(   1  -1 ) (   2   9)   -0.3046610277494683D-08
sin(   1  -1 ) (   2  11)   -0.3153475255761168D-08
sin(   1  -1 ) (   2  13)   -0.2592612989307051D-08
sin(   1  -1 ) (   3   2)   -0.3105516299092689D-10
sin(   1  -1 ) (   3   4)    0.2282507623146730D-09
sin(   1  -1 ) (   3   6)    0.1250182442738840D-08
sin(   1  -1 ) (   3   8)    0.3884706074135672D-08
sin(   1  -1 ) (   3  10)    0.2919145607257011D-08
sin(   1  -1 ) (   3  12)    0.9403926837108905D-09
sin(   1  -1 ) (   4   3)   -0.2164076283812181D-09
sin(   1  -1 ) (   4   5)   -0.8649463976996244D-09
sin(   1  -1 ) (   4   7)   -0.9908666305203136D-09
sin(   1  -1 ) (   4   9)    0.1271087732101629D-08
sin(   1  -1 ) (   4  11)    0.3584122562241348D-08
sin(   1  -1 ) (   5   2)    0.7837266791380954D-11
sin(   1  -1 ) (   5   4)    0.1350829773908567D-09
sin(   1  -1 ) (   5   6)    0.1464302680790337D-08
```

```
sin(   1  -1 ) (   5   8)    0.1453237588146330D-08
sin(   1  -1 ) (   5  10)   -0.1652265781571743D-08
sin(   1  -1 ) (   6   1)    0.8747904678072307D-11
sin(   1  -1 ) (   6   3)    0.7846341007140207D-10
sin(   1  -1 ) (   6   5)    0.7314728555020918D-09
sin(   1  -1 ) (   6   7)    0.2477471854879612D-08
sin(   1  -1 ) (   6   9)    0.5704798014628243D-08
sin(   1  -1 ) (   7   2)   -0.2816194835851615D-10
sin(   1  -1 ) (   7   4)   -0.4815263475646024D-09
sin(   1  -1 ) (   7   6)   -0.2552461056401023D-08
sin(   1  -1 ) (   7   8)   -0.7098456677713187D-08
sin(   1  -1 ) (   8   1)    0.1499291610334273D-11
sin(   1  -1 ) (   8   3)    0.9261902024518462D-11
sin(   1  -1 ) (   8   5)    0.1894710555582157D-12
sin(   1  -1 ) (   8   7)    0.1638415335667082D-09
sin(   1  -1 ) (   9   2)    0.2321264873061963D-11
sin(   1  -1 ) (   9   4)    0.6193903019997141D-11
sin(   1  -1 ) (   9   6)   -0.2017467018762032D-09
sin(   1  -1 ) (  10   1)   -0.2707712727807498D-11
sin(   1  -1 ) (  10   3)   -0.1397271974196371D-11
sin(   1  -1 ) (  10   5)   -0.6387660690592208D-10
sin(   1  -1 ) (  11   2)    0.1198270132498605D-11
sin(   1  -1 ) (  11   4)    0.2270558152963764D-10
sin(   1  -1 ) (  12   1)   -0.5112702869926338D-12
sin(   1  -1 ) (  12   3)    0.1276088160851436D-11
sin(   1  -1 ) (  13   2)   -0.2552176321702871D-12
sin(   2  -2 ) (   2   3)    0.6340790278993960D-10
sin(   2  -2 ) (   2   5)    0.5159003141427130D-09
sin(   2  -2 ) (   2   7)    0.2140769728198231D-08
sin(   2  -2 ) (   2   9)    0.3298122914393586D-08
sin(   2  -2 ) (   2  11)    0.6279296243986885D-08
sin(   2  -2 ) (   2  13)    0.7036627583920597D-08
sin(   2  -2 ) (   3   2)    0.5665096453166332D-10
sin(   2  -2 ) (   3   4)    0.1008842008799075D-09
sin(   2  -2 ) (   3   6)    0.3766836284934961D-10
sin(   2  -2 ) (   3   8)    0.3723501167088738D-09
sin(   2  -2 ) (   3  10)   -0.9030030618868948D-09
sin(   2  -2 ) (   3  12)   -0.1282015764925400D-08
sin(   2  -2 ) (   4   3)    0.7993172073003958D-10
sin(   2  -2 ) (   4   5)    0.9569415806452997D-09
sin(   2  -2 ) (   4   7)    0.1372306896947147D-08
sin(   2  -2 ) (   4   9)    0.5489764676700338D-08
sin(   2  -2 ) (   4  11)    0.8964754797034057D-08
sin(   2  -2 ) (   5   2)   -0.9235905865847337D-11
sin(   2  -2 ) (   5   4)   -0.2981366501164871D-10
sin(   2  -2 ) (   5   6)   -0.4038625522662028D-09
sin(   2  -2 ) (   5   8)   -0.3701993358721746D-08
sin(   2  -2 ) (   5  10)   -0.8040961437564745D-08
sin(   2  -2 ) (   6   3)    0.1106684717201655D-09
sin(   2  -2 ) (   6   5)    0.1003470039055837D-08
sin(   2  -2 ) (   6   7)    0.4052656594957586D-08
sin(   2  -2 ) (   6   9)    0.1072399595488710D-07
sin(   2  -2 ) (   7   2)    0.2513523319810928D-11
sin(   2  -2 ) (   7   4)   -0.3653207490878196D-10
sin(   2  -2 ) (   7   6)   -0.5336854493752702D-09
sin(   2  -2 ) (   7   8)   -0.2531981562783710D-08
sin(   2  -2 ) (   8   3)    0.2751033914649566D-10
```

```
sin(   2  -2 ) (   8   5)    0.5075050351757131D-09
sin(   2  -2 ) (   8   7)    0.2898169956424058D-08
sin(   2  -2 ) (   9   2)    0.3716768357702092D-11
sin(   2  -2 ) (   9   4)    0.4758094698744909D-10
sin(   2  -2 ) (   9   6)    0.2397449209420164D-09
sin(   2  -2 ) (  10   3)   -0.1514766470160853D-10
sin(   2  -2 ) (  10   5)   -0.9753528446825102D-10
sin(   2  -2 ) (  11   2)    0.3861587668889461D-12
sin(   2  -2 ) (  11   4)    0.4660682962472884D-11
sin(   3  -3 ) (   3   4)   -0.4488620239521181D-10
sin(   3  -3 ) (   3   6)   -0.5511201021948896D-09
sin(   3  -3 ) (   3   8)   -0.1930110214503018D-08
sin(   3  -3 ) (   3  10)   -0.6668508125145970D-08
sin(   3  -3 ) (   3  12)   -0.9476170486059573D-08
sin(   3  -3 ) (   4   3)   -0.2377844762056324D-10
sin(   3  -3 ) (   4   5)   -0.3007892979477410D-10
sin(   3  -3 ) (   4   7)    0.2449987991488348D-09
sin(   3  -3 ) (   4   9)    0.2493383349969122D-08
sin(   3  -3 ) (   4  11)    0.3871811572531579D-08
sin(   3  -3 ) (   5   4)   -0.9893563013919790D-10
sin(   3  -3 ) (   5   6)   -0.8908037118622097D-09
sin(   3  -3 ) (   5   8)   -0.4828837770280241D-08
sin(   3  -3 ) (   5  10)   -0.9660483592982481D-08
sin(   3  -3 ) (   6   5)    0.4918327072583328D-10
sin(   3  -3 ) (   6   7)    0.1117415118134875D-08
sin(   3  -3 ) (   6   9)    0.3603843372740518D-08
sin(   3  -3 ) (   7   4)   -0.1078605907078340D-09
sin(   3  -3 ) (   7   6)   -0.1475957763597489D-08
sin(   3  -3 ) (   7   8)   -0.5448534858673030D-08
sin(   3  -3 ) (   8   3)   -0.5045811955649109D-11
sin(   3  -3 ) (   8   5)   -0.4806642007284035D-10
sin(   3  -3 ) (   8   7)    0.1893439972727318D-09
sin(   3  -3 ) (   9   4)   -0.1826145489485322D-10
sin(   3  -3 ) (   9   6)   -0.1854215647076782D-09
sin(   3  -3 ) (  10   3)   -0.4102329717898472D-11
sin(   3  -3 ) (  10   5)   -0.5996788764457770D-10
sin(   3  -3 ) (  11   4)    0.3897078934047172D-11
sin(   3  -3 ) (  12   3)    0.2078386684346430D-12
sin(   4  -4 ) (   4   5)    0.3105143757457537D-10
sin(   4  -4 ) (   4   7)    0.2111597334119490D-09
sin(   4  -4 ) (   4   9)    0.1951342527266506D-08
sin(   4  -4 ) (   4  11)    0.3727391007169035D-08
sin(   4  -4 ) (   5   4)    0.1089213644685447D-10
sin(   4  -4 ) (   5   6)    0.8730034931990870D-11
sin(   4  -4 ) (   5   8)   -0.7730972749954444D-09
sin(   4  -4 ) (   5  10)   -0.1646424212850422D-08
sin(   4  -4 ) (   6   5)    0.5412343395056112D-10
sin(   4  -4 ) (   6   7)    0.1100461043171245D-08
sin(   4  -4 ) (   6   9)    0.3382309043870024D-08
sin(   4  -4 ) (   7   4)    0.2836794392341400D-11
sin(   4  -4 ) (   7   6)   -0.5718952789360153D-10
sin(   4  -4 ) (   7   8)   -0.5101109579039702D-09
sin(   4  -4 ) (   8   5)    0.1030759539879231D-09
sin(   4  -4 ) (   8   7)    0.7231397462334100D-09
sin(   4  -4 ) (   9   4)    0.9026067837820321D-11
sin(   4  -4 ) (   9   6)    0.2719969158439919D-10
sin(   4  -4 ) (  10   5)    0.8562603302069978D-11
```

```
sin(   4  -4 ) (  11   4)    0.3810440683908654D-11
sin(   5  -5 ) (   5   6)    0.1253884551499294D-11
sin(   5  -5 ) (   5   8)   -0.1626541738369835D-09
sin(   5  -5 ) (   5  10)   -0.4499169434704486D-09
sin(   5  -5 ) (   6   5)   -0.5690100710845616D-11
sin(   5  -5 ) (   6   7)    0.6188247938846168D-10
sin(   5  -5 ) (   6   9)    0.1996013260043539D-09
sin(   5  -5 ) (   7   6)   -0.6188247938846168D-10
sin(   5  -5 ) (   7   8)   -0.3992026520087077D-09
sin(   5  -5 ) (   8   5)   -0.4404361520553005D-11
sin(   5  -5 ) (   8   7)    0.1851729372787976D-10
sin(   5  -5 ) (   9   6)   -0.9258646863939880D-11
sin(   5  -5 ) (  10   5)   -0.3343716793718132D-11
```

Les résultats de la version vectorielle sont donnés ci-dessous.

```
     SERIES     Poisson Brackets         0
      NUMBER OF TERMS :      0




     SERIES     Poisson Brackets         1
      NUMBER OF TERMS :      0




     SERIES     Poisson Brackets         2
      NUMBER OF TERMS :      0




     SERIES     Poisson Brackets         3
      NUMBER OF TERMS :      1

     p1  p2      E1  E2             COEF
sin(   1  -1 ) (   1   2)   -0.5215977519644021D-10


     SERIES     Poisson Brackets         4
      NUMBER OF TERMS :      0
```

```
     SERIES     Poisson Brackets         5
      NUMBER OF TERMS :      5


     p1  p2       E1  E2             COEF
sin(   1  -1 ) (   1   4)   -0.1391947476901391D-09
sin(   1  -1 ) (   2   3)   -0.1794043464677527D-09
sin(   1  -1 ) (   3   2)   -0.3105516299092689D-10
sin(   2  -2 ) (   2   3)    0.6340790278993960D-10
sin(   2  -2 ) (   3   2)    0.5665096453166332D-10


     SERIES     Poisson Brackets         6
      NUMBER OF TERMS :      0




     SERIES     Poisson Brackets         7
      NUMBER OF TERMS :     12


     p1  p2       E1  E2             COEF
sin(   1  -1 ) (   1   6)   -0.2725534073118022D-09
sin(   1  -1 ) (   2   5)   -0.6954622817661499D-09
sin(   1  -1 ) (   3   4)    0.2282507623146730D-09
sin(   1  -1 ) (   4   3)   -0.2164076283812181D-09
sin(   1  -1 ) (   5   2)    0.7837266791380954D-11
sin(   1  -1 ) (   6   1)    0.8747904678072307D-11
sin(   2  -2 ) (   2   5)    0.5159003141427130D-09
sin(   2  -2 ) (   3   4)    0.1008842008799075D-09
sin(   2  -2 ) (   4   3)    0.7993172073003961D-10
sin(   2  -2 ) (   5   2)   -0.9235905865847337D-11
sin(   3  -3 ) (   3   4)   -0.4488620239521181D-10
sin(   3  -3 ) (   4   3)   -0.2377844762056324D-10


     SERIES     Poisson Brackets         8
      NUMBER OF TERMS :      0




     SERIES     Poisson Brackets         9
      NUMBER OF TERMS :     19


     p1  p2       E1  E2             COEF
sin(   1  -1 ) (   1   8)   -0.4969922086550716D-09
sin(   1  -1 ) (   2   7)   -0.1823677094096726D-08
sin(   1  -1 ) (   3   6)    0.1250182442738840D-08
sin(   1  -1 ) (   4   5)   -0.8649463976996244D-09
```

```
sin(   1  -1 ) (   5   4)    0.1350829773908567D-09
sin(   1  -1 ) (   6   3)    0.7846341007140207D-10
sin(   1  -1 ) (   7   2)   -0.2816194835851615D-10
sin(   1  -1 ) (   8   1)    0.1499291610334273D-11
sin(   2  -2 ) (   2   7)    0.2140769728198231D-08
sin(   2  -2 ) (   3   6)    0.3766836284934964D-10
sin(   2  -2 ) (   4   5)    0.9569415806452997D-09
sin(   2  -2 ) (   5   4)   -0.2981366501164871D-10
sin(   2  -2 ) (   6   3)    0.1106684717201655D-09
sin(   2  -2 ) (   7   2)    0.2513523319810929D-11
sin(   3  -3 ) (   3   6)   -0.5511201021948896D-09
sin(   3  -3 ) (   4   5)   -0.3007892979477410D-10
sin(   3  -3 ) (   5   4)   -0.9893563013919790D-10
sin(   4  -4 ) (   4   5)    0.3105143757457537D-10
sin(   4  -4 ) (   5   4)    0.1089213644685447D-10


     SERIES     Poisson Brackets        10
      NUMBER OF TERMS :      0




     SERIES     Poisson Brackets        11
      NUMBER OF TERMS :     29


     p1  p2       E1  E2             COEF
sin(   1  -1 ) (   2   9)   -0.3046610277494682D-08
sin(   1  -1 ) (   3   8)    0.3884706074135672D-08
sin(   1  -1 ) (   4   7)   -0.9908666305203136D-09
sin(   1  -1 ) (   5   6)    0.1464302680790337D-08
sin(   1  -1 ) (   6   5)    0.7314728555020918D-09
sin(   1  -1 ) (   7   4)   -0.4815263475646024D-09
sin(   1  -1 ) (   8   3)    0.9261902024518462D-11
sin(   1  -1 ) (   9   2)    0.2321264873061963D-11
sin(   1  -1 ) (  10   1)   -0.2707712727807498D-11
sin(   2  -2 ) (   2   9)    0.3298122914393586D-08
sin(   2  -2 ) (   3   8)    0.3723501167088738D-09
sin(   2  -2 ) (   4   7)    0.1372306896947147D-08
sin(   2  -2 ) (   5   6)   -0.4038625522662028D-09
sin(   2  -2 ) (   6   5)    0.1003470039055837D-08
sin(   2  -2 ) (   7   4)   -0.3653207490878196D-10
sin(   2  -2 ) (   8   3)    0.2751033914649566D-10
sin(   2  -2 ) (   9   2)    0.3716768357702092D-11
sin(   3  -3 ) (   3   8)   -0.1930110214503018D-08
sin(   3  -3 ) (   4   7)    0.2449987991488348D-09
sin(   3  -3 ) (   5   6)   -0.8908037118622097D-09
sin(   3  -3 ) (   6   5)    0.4918327072583328D-10
sin(   3  -3 ) (   7   4)   -0.1078605907078340D-09
sin(   3  -3 ) (   8   3)   -0.5045811955649109D-11
sin(   4  -4 ) (   4   7)    0.2111597334119490D-09
sin(   4  -4 ) (   5   6)    0.8730034931990870D-11
sin(   4  -4 ) (   6   5)    0.5412343395056112D-10
sin(   4  -4 ) (   7   4)    0.2836794392341400D-11
```

```
sin(   5  -5 ) (   5   6)    0.1253884551499294D-11
sin(   5  -5 ) (   6   5)   -0.5690100710845616D-11


     SERIES     Poisson Brackets        12
      NUMBER OF TERMS :      0




     SERIES     Poisson Brackets        13
      NUMBER OF TERMS :     39


     p1  p2       E1  E2             COEF
sin(   1  -1 ) (   2  11)   -0.3153475255761168D-08
sin(   1  -1 ) (   3  10)    0.2919145607257011D-08
sin(   1  -1 ) (   4   9)    0.1271087732101629D-08
sin(   1  -1 ) (   5   8)    0.1453237588146329D-08
sin(   1  -1 ) (   6   7)    0.2477471854879612D-08
sin(   1  -1 ) (   7   6)   -0.2552461056401023D-08
sin(   1  -1 ) (   8   5)    0.1894710555582157D-12
sin(   1  -1 ) (   9   4)    0.6193903019997141D-11
sin(   1  -1 ) (  10   3)   -0.1397271974196371D-11
sin(   1  -1 ) (  11   2)    0.1198270132498605D-11
sin(   1  -1 ) (  12   1)   -0.5112702869926338D-12
sin(   2  -2 ) (   2  11)    0.6279296243986885D-08
sin(   2  -2 ) (   3  10)   -0.9030030618868948D-09
sin(   2  -2 ) (   4   9)    0.5489764676700338D-08
sin(   2  -2 ) (   5   8)   -0.3701993358721746D-08
sin(   2  -2 ) (   6   7)    0.4052656594957586D-08
sin(   2  -2 ) (   7   6)   -0.5336854493752702D-09
sin(   2  -2 ) (   8   5)    0.5075050351757131D-09
sin(   2  -2 ) (   9   4)    0.4758094698744909D-10
sin(   2  -2 ) (  10   3)   -0.1514766470160853D-10
sin(   2  -2 ) (  11   2)    0.3861587668889461D-12
sin(   3  -3 ) (   3  10)   -0.6668508125145967D-08
sin(   3  -3 ) (   4   9)    0.2493383349969122D-08
sin(   3  -3 ) (   5   8)   -0.4828837770280243D-08
sin(   3  -3 ) (   6   7)    0.1117415118134875D-08
sin(   3  -3 ) (   7   6)   -0.1475957763597489D-08
sin(   3  -3 ) (   8   5)   -0.4806642007284035D-10
sin(   3  -3 ) (   9   4)   -0.1826145489485322D-10
sin(   3  -3 ) (  10   3)   -0.4102329717898472D-11
sin(   4  -4 ) (   4   9)    0.1951342527266506D-08
sin(   4  -4 ) (   5   8)   -0.7730972749954444D-09
sin(   4  -4 ) (   6   7)    0.1100461043171245D-08
sin(   4  -4 ) (   7   6)   -0.5718952789360153D-10
sin(   4  -4 ) (   8   5)    0.1030759539879231D-09
sin(   4  -4 ) (   9   4)    0.9026067837820321D-11
sin(   5  -5 ) (   5   8)   -0.1626541738369835D-09
sin(   5  -5 ) (   6   7)    0.6188247938846168D-10
sin(   5  -5 ) (   7   6)   -0.6188247938846168D-10
sin(   5  -5 ) (   8   5)   -0.4404361520553005D-11
```

```
     SERIES     Poisson Brackets        14
      NUMBER OF TERMS :      0




     SERIES     Poisson Brackets        15
      NUMBER OF TERMS :     46


     p1  p2       E1  E2             COEF
sin(   1  -1 ) (   2  13)   -0.2592612989307051D-08
sin(   1  -1 ) (   3  12)    0.9403926837108905D-09
sin(   1  -1 ) (   4  11)    0.3584122562241348D-08
sin(   1  -1 ) (   5  10)   -0.1652265781571743D-08
sin(   1  -1 ) (   6   9)    0.5704798014628243D-08
sin(   1  -1 ) (   7   8)   -0.7098456677713187D-08
sin(   1  -1 ) (   8   7)    0.1638415335667082D-09
sin(   1  -1 ) (   9   6)   -0.2017467018762032D-09
sin(   1  -1 ) (  10   5)   -0.6387660690592208D-10
sin(   1  -1 ) (  11   4)    0.2270558152963764D-10
sin(   1  -1 ) (  12   3)    0.1276088160851436D-11
sin(   1  -1 ) (  13   2)   -0.2552176321702871D-12
sin(   2  -2 ) (   2  13)    0.7036627583920597D-08
sin(   2  -2 ) (   3  12)   -0.1282015764925400D-08
sin(   2  -2 ) (   4  11)    0.8964754797034057D-08
sin(   2  -2 ) (   5  10)   -0.8040961437564745D-08
sin(   2  -2 ) (   6   9)    0.1072399595488710D-07
sin(   2  -2 ) (   7   8)   -0.2531981562783710D-08
sin(   2  -2 ) (   8   7)    0.2898169956424058D-08
sin(   2  -2 ) (   9   6)    0.2397449209420164D-09
sin(   2  -2 ) (  10   5)   -0.9753528446825102D-10
sin(   2  -2 ) (  11   4)    0.4660682962472884D-11
sin(   3  -3 ) (   3  12)   -0.9476170486059573D-08
sin(   3  -3 ) (   4  11)    0.3871811572531579D-08
sin(   3  -3 ) (   5  10)   -0.9660483592982481D-08
sin(   3  -3 ) (   6   9)    0.3603843372740518D-08
sin(   3  -3 ) (   7   8)   -0.5448534858673030D-08
sin(   3  -3 ) (   8   7)    0.1893439972727318D-09
sin(   3  -3 ) (   9   6)   -0.1854215647076782D-09
sin(   3  -3 ) (  10   5)   -0.5996788764457770D-10
sin(   3  -3 ) (  11   4)    0.3897078934047172D-11
sin(   3  -3 ) (  12   3)    0.2078386684346430D-12
sin(   4  -4 ) (   4  11)    0.3727391007169035D-08
sin(   4  -4 ) (   5  10)   -0.1646424212850422D-08
sin(   4  -4 ) (   6   9)    0.3382309043870024D-08
sin(   4  -4 ) (   7   8)   -0.5101109579039702D-09
sin(   4  -4 ) (   8   7)    0.7231397462334100D-09
sin(   4  -4 ) (   9   6)    0.2719969158439919D-10
sin(   4  -4 ) (  10   5)    0.8562603302069978D-11
sin(   4  -4 ) (  11   4)    0.3810440683908654D-11
sin(   5  -5 ) (   5  10)   -0.4499169434704486D-09
sin(   5  -5 ) (   6   9)    0.1996013260043539D-09
sin(   5  -5 ) (   7   8)   -0.3992026520087077D-09
```

```
sin(   5  -5 ) (   8   7)    0.1851729372787976D-10
sin(   5  -5 ) (   9   6)   -0.9258646863939880D-11
sin(   5  -5 ) (  10   5)   -0.3343716793718132D-11


     SERIES     Poisson Brackets        16
      NUMBER OF TERMS :      0
```

Bibliographie

[Abu-el-Ata et Chapront, 1975] Abu-el-Ata, N., Chapront, J., 1975. Développements analytiques de l'inverse de la distance en mécanique céleste, *Astronomy and Astrophysics*, 38, 57-66.

[Brouwer & Clemence, 1961] Brouwer, D., Clemence, G.M., 1961. *Methods of Celestial Mechanics*, Academic Press, New York & London.

[Ellis et Murray] Ellis, Keren M., Murray, D., 2000. The disturbing function in Solar system dynamics, *Icarus*, 147, 129-144.

[Aide MSNam] Henrard, J., 2004. Summary of the MSNam, FUNDP, Namur.

[Giorgilli, 2009] Giorgilli, A., 2009, Notes du cours de Mécanique Céleste, *Dipartimento di Matematica, UNIMI*, 2009.

[Henrard, 1989] Henrard, J., 1989. A survey of Poisson series Processors, *Celestial Mechanics and Dynamical Astronomy 45*, 245-253.

[Henrard, 2000] Henrard, J., 2000. A note on a general algorithm for two-body expansions, *Celestial Mechanics and Dynamical Astronomy 76*, 283-289.

[Henrard, 2006] Henrard, Jacques, 2006. Notes de cours de théorie des perturbations, *Département de Mathématique, FUNDP*, 2006.

[Knuth, 1973] Knuth, D.E., 1973. *The Art of Computer Programming*, Addison Wesley, Boston, USA.

[Laskar, 1988] Laskar, J., 1988. Les variables de Poincaré et le développement de la fonction perturbatrice, *Groupe de travail sur la lecture des Méthodes Nouvelles de la Mécanique Céleste*, SCMC du Bureau des Longitudes, Paris, 1-25.

[Libert, 2007] Libert, A.-S., 2007. Dynamique séculaire du problème des trois corps appliqué aux systèmes extrasolaires, thèse de doctorat, Presses Universitaires de Namur, FUNDP, Namur, 23-29.

[Murray et Dermott, 1999] Murray, C.D., Dermott, S.F., 1999. *Solar system dynamics*, Cambridge University Press, Cambridge, UK.